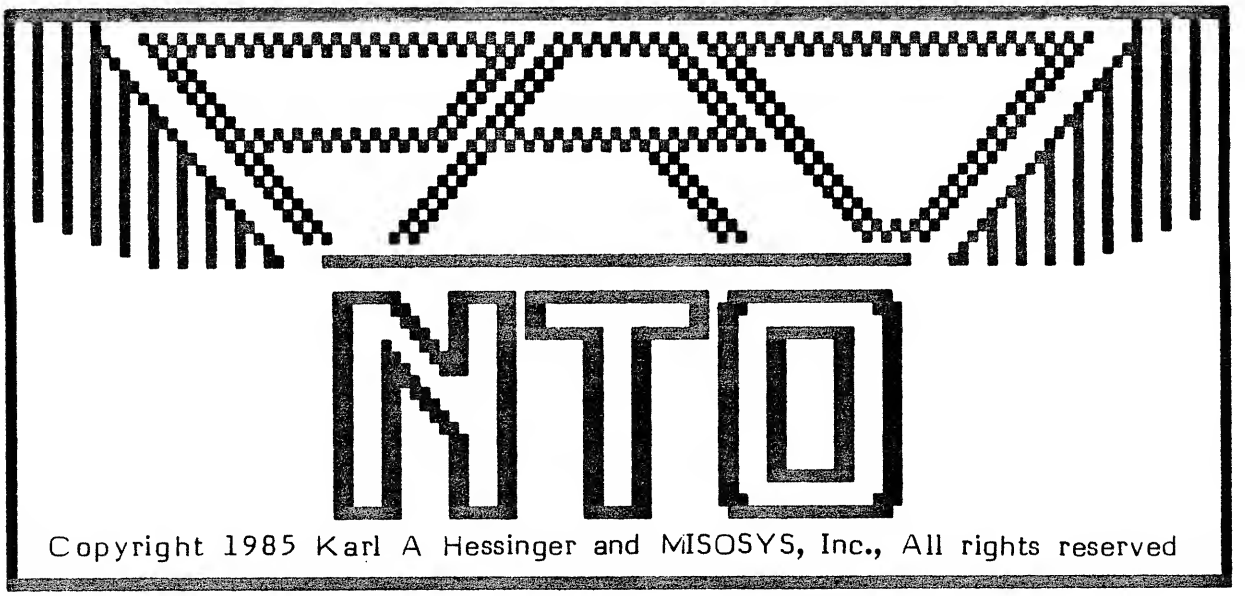



```
|                                                                                |
|  Wh+-----------------------------------------+                                 |
|    |                +----------------------------------------+                 |
|    |   Apr  1, 1985|     Aside from being one of the most   |s to come along  |
| for|   +----+----+-|significant new products to +-------------------------+   |
| win|   | Su | Mo | |for the 128K TRS-80 Model 4/| Display [ 171.5650    ] |   |
| por|   +----+----+-|PRO-NTO happens to be a powe|                          |f  |
| the|   |    |< 1>| |display window controller an| Memory  [ 123.7650    ] |d  |
| at |   +----+----+-|application manager facility|                          |   |
|    |   |  7 |  8 | |therefore, supports two prim|     * : Mul    7 8 9     |   |
|    |   +----+----+-|functions: the control of th|     /   Div    4 5 6     |.  |
| col|   | 14 | 15 | |screen or portions of the sc|     -   Sub    1 2 3     |0  |
| col|   +----+----+-|term a "window", and a manag|     + ; Add    0 . =     |   |
| ate|   | 21 | 22 | |applications programs invoke|                          |   |
| pro|   +----+----+-|touch of a button.          |   DNARW=STO  UPARW=RCL   |   |
| int|   | 28 | 29 | |---------------------------+-------------------------+   |
| for|   +----+----+-|[NOTES   ] [04/01/85 09:49:18] [    6]| so that you may  |
| eas|   |    |    | |                                    |O-NTO also in-    |
| clu|   +----+----+-|  Add  Card  Delete  Edit  First  Goto  |use in writing    |
| pro|                |  Id   Last  Next    Prev  Search     _ |                  |
|    |           Goto+-----------------------------------------+                  |
|    +-----------------------------------------+ position, the current display |
| screen image as well as the library overlay region of the DOS is saved in one  |
|                                                                                |
```

* * * NOTICE * * *

* * * WARRANTY * * *

This software program(s) is warranted to perform as documented when used on the specified hardware operating under the specified disk operating system as shown on the accompanying documentation. If within 90 days of the date of purchase the program is found to be defective due to a bug in the code, the publisher will, upon request, provide a patch to correct the bug or will update the program diskette with a corrected copy within a reasonable time period after return of the program diskette to the publisher. If within 90 days of the date of purchase the documentation proves defective due to missing pages, the publisher will provide substitutes for the missing pages upon request.

The publisher shall have no liability or responsibility to the purchaser or any other person, company, or entity with respect to any liability, loss, or damage caused or alleged to have been caused by this product, including but not limited to any interruption of service, loss of business and anticipatory profits, or consequential damages resulting from the operation or use of this program.

* * * ATTENTION * * *

MISOSYS,Inc.
P.O. Box 239
Sterling, Virginia 22170-0239
703-450-4181

Talking about applications touches on the other aspect of PRO-NTO - the application manager. PRO-NTO makes use of the function keys on the keyboard of your computer. By using the <F1>, <F2> and <F3> keys in both their unshifted and shifted positions, PRO-NTO provides keystroke invocation of six applications. The function keys are not "taken over" by PRO-NTO until you activate the application manager via a <CONTROL-P> key request [you have the opportunity of specifying something other than <CONTROL-P> to activate PRO-NTO when it is installed]. Applications are small programs which are designed to run entirely within the library overlay region established by the DOS. When PRO-NTO is installed, four out of the many applications provided with PRO-NTO are loaded into the alternate RAM bank set aside for PRO-NTO's use. Each of those four applications are assigned to one of the six function key positions. Two additional applications which are internal functions of PRO-NTO are assigned to the remaining two function key positions. These are the LIBrary EXECutive (LIBEXEC) which allows you to invoke a DOS LIBRARY command and the UNIVERSAL function which permits you to invoke applications directly from disk.

Regardless of what program is currently being run on your machine, if it requests a key entry from the standard DOS @KEY SuperVisor Call facility, you will be able to temporarily invoke PRO-NTO and request one of the six memory resident applications (or additional ones from disk). When you are finished with that application, you will be returned to the program you were running just as if you never interrupted it. There may be a few exceptions to programs which are incompatible with PRO-NTO. For instance, if you are receiving data from the communications line while running COMM and interrupt it to invoke a PRO-NTO application, it's capture buffer may overflow while you are using the application [see the note on using DIALER with COMM in the section documenting the DIALER application]. There are also a few programs known to not use the standard DOS @KEY SVC for their keyboard input. One is SuperSCRIPSIT sold by Tandy Corp. These two examples highlight a few known programs which either cannot be interrupted by PRO-NTO or should not be.

In order to gain access to the window controller and application manager, PRO-NTO must be installed in your system. If you are going to write or use BASIC language programs which access the window controller, the WINLINK device driver must also be installed. If you would like to establish a Job Control Language procedure to load PRO-NTO and automatically invoke an application - regardless of whether or not the application you desire is one of the four memory resident applications, see the information on PRUN. Once PRO-NTO is installed, a sophisticated and powerful adjunct to your computer is at your fingertips.

BACKUP before you begin

All of the PRO-NTO files noted in the appendix have been provided on a single 40-track double density data diskette. This is not a protected disk! Please, before you go any further in doing anything with the diskette supplied, make a BACKUP. Do this in a two-drive system by placing a blank diskette in your top drive and issuing the command,

```
FORMAT :1 (NAME="PRONTO",DDEN,CYL=40)
```

Then issue the command,

```
BACKUP :0 :1 (X)
```

When the system prompts you for the SOURCE disk, remove your SYSTEM disk from the bottom drive and insert the PRONTO diskette. Follow the procedures for BACKUP in your DOS manual. When you have completed the BACKUP, keep your PRONTO disk in a safe place - it is valuable. Use your BACKUP copy as a working disk. It may be prudent to make a few copies for your own use.

Modifying PRONTO/CMD

PRO-NTO requires approximately 2.5K of high memory RAM, a small segment of low memory in the DOS driver region, one 32-K RAM bank, and TRSDOS 6.2 or its equivalent. As noted previously, PRO-NTO will load four application programs from disk. Four of the many application programs which are provided with PRO-NTO [these can be identified in the table of contents] have been selected for you with your purchase of PRO-NTO. These are: AFPCALC, CALendar, CARD, and DIALER. You are not limited to loading only these four applications. A program called DEFAULTS/CMD is provided so that you can change the four applications loaded by PRO-NTO to those of your choosing. Once you gain familiarity with each of the applications provided with PRO-NTO, you will be in a good position to select which ones you want immediately available at the touch of a function key button. Any other application which is not installed into memory can still be invoked directly from disk from either the PRO-NTO menu via the "UNIVERSAL application invoker" or by using the PRUN facility at "DOS Ready".

If you so desire to change the applications installed with PRO-NTO, invoke the modification program via the command:

```
DEFAULTS
```

You will be prompted for each of the four application's to be loaded by PRO-NTO. These will be identified by function key number [F1, F2, sF1, sF2] along with the current application assigned to the key. All you need do is respond with <ENTER> to select the default, or type in another application's filename (note that you are supposed to enter neither the file extension nor the drive specification).

Installing PRO-NTO

When you have finished customizing PRO-NTO's assigned applications, you can install the window controller, the application manager, and the four PRO-NTO selected applications simply by entering the command:

PRONTO (Active=ddd)

The parameter, "Active=ddd", is optional. It is used to specify the character code assigned to activate PRO-NTO after installation. If you do not enter the parameter, the default activation code will be 16 (decimal) which is entered from the keyboard as <CONTROL-P>. You can alter the default activation character by changing it with DEFAULTS. You can also enter your selection when you install PRO-NTO via the command noted above. If you choose to enter the parameter, the "ddd" field represents your character code value which must be entered in either decimal format [ddd] or hexadecimal format [X'xx'].

As noted previously, PRO-NTO requires certain physical resources from the system. First, PRO-NTO requires DOS version 6.2 or later. If PRO-NTO detects that you are trying to install it on an earlier version, it will display the error message,

Can't install on DOS Version x.y

and abort your request. PRO-NTO behaves like a filter and is assigned the device name, "*PO"; thus, it needs a system Device Control Block field (DCB). Before requesting a spare DCB to use for its assignment, PRO-NTO will check to make sure it has not already been installed. If it has, there is no need to install it again and it will post the error message,

The *PO device is already in use!

and abort your request. If a spare DCB field can not be obtained from the system, PRO-NTO advises you of this by posting the message,

No system DCB is available for *PO!

and, again, your request will not be fulfilled. PRO-NTO requires the usage of a high memory block of RAM and a small low memory block of RAM. If either of these is unobtainable, PRO-NTO will not be able to install itself and will advise you of this situation by posting one of the following two messages:

Can't alter high memory pointer!
No low memory is available for PRO-NTO!

PRO-NTO also requires one 32K external memory bank for storage of the four applications and the window control features. It will search for one which is available for use. If none can be found, PRO-NTO will post the error message,

No external memory is available for PRO-NTO!

and abort the request.

Since all of the system resources have now been obtained, PRO-NTO will install itself and load the four applications that have been assigned by the DEFAULTS modification program, or the default applications if you have not selected your own group of four. If you do not specify an activation character via the "ACTIVE=ddd" parameter, PRO-NTO defaults to <CONTROL-P>. Thereafter, PRONTO is invoked simply by depressing the <CONTROL> and <P> keys simultaneously (or other keystroke for which you have specified during the installation by either the ACTIVE parameter or via DEFAULTS). Something similar to the following will be displayed as PRO-NTO installs itself and the four designated applications into memory:

```
PRO-NTO - Window and Application Manager - Version 1.0a
Copyright (c) 1985, Karl A. Hessinger, MicroConsultants-East

Activation character = <Control><P>

Reading application : Calendar
Reading application : Calculator
Reading application : Dialer
Reading application : Card Filer

Pronto installation complete
```

The astute sophisticated DOS 6 user will have noted by now that although PRO-NTO has been stated to be a filter, you have not been required to enter the SET command nor the FILTER command. That's because PRO-NTO has performed those functions for you. PRO-NTO has essentially implemented the following:

```
SET *PO PRONTO
FILTER *KI *PO
```

in order to streamline and simplify the installation procedure, thereby saving you from these entries. There is another point which must be discussed. The sophisticated DOS user has been known to SYSGEN all sorts of things for automatic loading when you BOOT your system. PRO-NTO may NOT be one of those things SYSGENed by the DOS SYSGEN command. If you wish to have it available when you BOOT your machine, it is suggested that you either AUTO it or provide it as part of a BOOT Job Control Language file. PRO-NTO may, however, be removed from the system whenever it has been the last module installed into both high and low memory by specifying the following command:

```
PRONTO (REMOVE)
```

In general, the closing parenthesis is not required. Also, the parameter, "REMOVE", may be abbreviated to the single letter, "R". The word, "OFF", may also be entered in lieu of "REMOVE" or "R". If you enter this parameter incorrectly, or enter some other un-supported parameter, the following error message will be displayed and the program will abort.

Parameter error!

If PRO-NTO finds that you have requested its removal but it is nowhere to be found in memory, it will inform you of this dilemma by posting the message,

PRO-NTO is not installed!

In order to be able to remove itself once it finds itself in memory, PRO-NTO must have been the most recent module loaded into both high memory and low memory. If some other module has been installed after PRO-NTO, it will not be able to return the memory to the system, so it will not remove itself. This unfortunate situation is revealed by the message,

Can't reclaim memory - PRO-NTO not removed!

When PRO-NTO does not have these restrictions, it will return the system resources to the system including the low and high memory, the system DCB, and the external memory bank. The informative message,

PRO-NTO is now removed

will be posted to inform you of this operation.

Operation Keystrokes

If an application supports the functions of import or export, they may be requested by depressing <CLEAR> <LEFT ARROW> or <CLEAR> <RIGHT ARROW> respectively. Furthermore, any of the supplied applications excluding TERM can be exited by depressing <BREAK>.

The command sets in all of the PRO-NTO applications have been carefully selected to provide a uniform set of command words throughout the entire PRO-NTO package. For instance, NEXT and PREVIOUS will always mean the same thing in each application. Thus you will probably not find yourselves getting confused as to which command letters do what in an application. In addition, each application presents a brief menu of the commands supported by it. In this way, you will find that the collection of applications provided with PRO-NTO form an integrated set of functions. With the capability of moving data easily between applications via the EXPORT and IMPORT functions, the applications work extremely well together.

Data EXPORT and IMPORT

The functions of data export and import are a significant feature of the PRO-NTO window controller. They permit you to mark a block of data on the screen and pass it to another application or program. The information which is marked appears to the receiving program exactly like it had been manually typed at the keyboard. What has been eliminated for you is the unnecessary drudgery of re-typing.

Although the two functions may be easily defined, it may require a little more explanation as to the distinction between export and import. We'll try to expand on the distinction with the following discussion. You have interrupted program ABC or application ABC to invoke application XYZ. Application XYZ is now running and is probably looking for some kind of input. If, by chance, a piece of data needed to be input into XYZ happened to be on the video screen from ABC, you can specify IMPORT. When the import function is requested, the previous screen will be temporarily displayed and you will be able to mark a rectangle of the screen. The marked rectangle may be a single line, a segment of a single line, or more than one line. When you have completed the marking, the XYZ screen will be restored and what you had marked on the ABC screen will be input automatically - one character at a time for each keyboard request.

For an example of this import function, say you are up in ADDRESS and want to dial a phone number stored in the DATA2 field. Since the autodialing function is in DIALER, you invoke DIALER (the XYZ) from ADDRESS (the ABC), go to MANUAL DIAL mode, then import the phone number into DIALER. When the dialing is complete, you exit DIALER and are returned to ADDRESS. For another example of import, see the comments in AFPCALC concerning the preparation of the AFPCALC documentation.

The function of export behaves similarly but is based on a reversal of the reference. You are in program ABC or application ABC awaiting input of some data. You invoke application XYZ to get a display of the data needed by ABC. Once the data is found, you request the export function which gives you the opportunity of marking a rectangle of the XYZ screen. Again, as in the case of import, the marked rectangle may be a single line, a segment of a single line, or more than one line; however, you will be confined to the area within the boundaries of the window. When you have completed the XYZ marking, the XYZ application will close its window and terminate passing control back to ABC. The ABC screen will be restored and what you had marked on the XYZ screen will be input to ABC automatically - one character at a time for each keyboard request.

When import or export has been requested, the data area to be transferred is conceptualized as a rectangle of space on the screen. In either case, there are two ways of controlling what PRO-NTO does at the end of each line imported or exported depending on how you mark the closure of the rectangle. If you close the rectangle via the <ENTER> key, a carriage return will be added to the "input" at the end of each marked line. This carriage return will be appended regardless of whether the marked rectangle is one or

more lines. If the rectangle is closed by the depression of <SHIFT> <ENTER>, then the line is input from the beginning mark to the ending mark in a continuous stream; no carriage returns are added by PRO-NTO.

The rectangle is defined by the two points making up its northwest to southeast diagonal. These two points may be marked in the following manner:

1. Position the cursor to the upper left corner of the rectangle which will contain the information. The four arrow keys, <LEFT>, <RIGHT>, <UP>, and <DOWN>, will move the cursor around the screen.

2. Depress <CONTROL><B> to mark the beginning of the rectangle block. The character under the cursor will be replaced on the screen with a left square bracket which indicates the marked position. Don't worry about the bracket displayed; the correct character will be provided as input.

3. Position the cursor to the lower right corner of this rectangle again using the four arrow keys. This position may be on the same display row as the beginning mark.

4. Depress the <ENTER> or <SHIFT> <ENTER> key to mark the end of the marked block. This now defines the rectangle. The export or import will commence. The export or import function may be aborted anytime prior to marking the end of the rectangular block simply by depressing the <BREAK> key.

Spend a little time practicing with export and import as the two functions are extremely powerful and quite useful. Besides saving you many thousands of keystrokes, they will also reduce the errors associated with manual typing of data input. You will also get used to the difference in behavior of the ported data stream depending on the key code used to close the data rectangle. The keystroke(s) used will invariably depend on the expectations of the receiving program or application.

Invoking Applications

Any application may be invoked simply by first activating the PRO-NTO application manager via <CONTROL-P> (or other character which you have specified during the PRO-NTO installation). PRO-NTO will display a menu outlining the applications corresponding to each function key. The menu will look something like this depending on which applications have been installed:

```
+------------------------+
|   F1 - Calculator      |
|  sF1 - Calendar        |
|   F2 - Card Filer      |
|  sF2 - Bringup         |
|   F3 - Universal       |
|  sF3 - LIB Exec        |
+------------------------+
```

All you need do is then depress the FUNCTION KEY associated with the application as shown in the menu. The keys, <F3> and <SHIFT-F3> are always assigned respectively to the UNIVERSAL Application Invocation and LIBrary EXECutive functions. The remaining four keys [<F1>, <F2>, <sF1>, and <sF2>] may be reassigned by DEFAULTS. Any application which is not assigned to a function key may be invoked directly from disk via the UNIVERSAL application or by PRUN.

If an application supplied with PRO-NTO cannot open a window, or the maximum recursion level has been reached, a beep will be sounded from your computer's internal speaker to let you know that the keystroke was accepted but could not be acted upon. The beeping is controlled entirely by the application and is not an internal part of the window controller. In this way, applications have the capability of controlling the "noise" level of the error condition.

One last point. Once PRO-NTO has been activated, the three function keys and their shifted key codes are trapped by PRO-NTO for keystroke invocation of the six functions identified in the menu. Thus, no application can use a function key code for internal purposes as PRO-NTO will automatically invoke the corresponding application when the key is depressed - even if you are within an application! It is this feature which permits you to consecutively recurse through four applications.

CAUTION: When you are using any application which opens up a data file, do NOT remove the disk containing the data file until you have exited the application.

LIB Exec

This is more of an internal function of PRO-NTO as it is an application program. The rational behind LIB EXEC is this. The DOS provides a SuperVisor Call facility, called @CMNDR, to invoke any command from another program and return to the program when the invoked command completes. Since programs utilizing this facilty may want to restrict the "commands" to those which operate solely within the library overlay region of the DOS, the facility provides a method to restrict the invocation to DOS library commands only. That's why, for instance, the facility used by BASIC in its "SYSTEM [command]" doesn't permit anything but library commands.

Now most commercial software programs do not provide access to this DOS facility. Either the programmers did not know how to do this or they didn't bother to concern themselves with giving you access to such things as REMOVE, DIR, FREE, FORMS, SETCOM, etc. Because the DOS library commands execute solely within the library overlay region - a region of memory also preserved by PRO-NTO prior to loading PRO-NTO applications, PRO-NTO can provide the DOS Command facility negleted by all of those other programs. PRO-NTO allows you to interrupt programs when they are waiting for a key entry. Thus, function key <sF3> has been provided to prompt you for a DOS library command for its invocation. Any DOS library command may be executed just as if you were at DOS Ready.

When you depress <SHIFT> <F3>, the display screen will clear and you will receive the prompt message,

```
Command?
```

Simply enter your desired DOS library command and it will be invoked. If you have forgotten which commands they are, just enter "LIB". The DOS LIB command shows you the names of the library commands. At the conclusion of the command's execution, you will again receive the "Command?" prompt so you can enter another command. When you are finished and want to return to the program which you interrupted, just depress the <BREAK> key. The display screen will be restored to its appearance prior to your interrupt and your program will continue.

Although the execution of non-library commands is restricted, any program which will execute entirely in the range of X'2600' through X'2FFF' [which is the DOS library overlay region] may be invoked using the RUN library command. Some of the programs available from MISOSYS which fall into this category are:

PRO-X-FTS	- X-modem protocol file transfer
PRO-HELP	- On-line help facility HELP files

and can be invoked from other programs via the LIB EXEC feature of PRO-NTO by specifying a command syntax of, for instance:

```
RUN HELPP(EXPORT)
```

LIB EXEC supports both import and export. For example, if you are in a program which is looking for a file specification, you could interrupt it at the point it is awaiting the filespec, activate PRO-NTO and invoke LIB EXEC, do a CAT command to display the files on a drive, then depress EXPORT to mark the filename to be automatically passed back to the program's input. Alternatively, if there is a command displayed on the program's screen, you can invoke LIB EXEC and IMPORT the command string from the program's display screen.

Universal

Since PRO-NTO supports only four resident applications, you will probably find some of the other applications so useful that you will also want to be able to access them. So long as you keep an applications' disk "on-line", you can invoke any application stored on that disk via the UNIVERSAL function assigned to <F3>. When you activate PRO-NTO and depress <F3>, a window will be opened up on the screen as follows:

At the "Application?" prompt, enter the name of the application file you wish to invoke. You can enter a drive specification, but omit the file extension. PRO-NTO will automatically add the "/APP" file extension used for application files and attempt to load the application from disk.

If PRO-NTO cannot locate the application requested, a short beep will be emitted from your computer's internal speaker and PRO-NTO will return to what program was interrupted after the display screen is restored. If you do not want to invoke an application after seeing the above window [i.e. you may have inadvertantly depressed <F3>], just depress the <BREAK> key and PRO-NTO will abort your request.

If you wish to import the application name for the prompt from the display screen in effect which you interrupted, just depress IMPORT when PRO-NTO is prompting you for the application's name. Remember, IMPORT is specified by simultaneous depress of <CLEAR> <LEFT ARROW>.

If you forget the name of the application which you want to invoke, why not just depress <SHIFT> <F3> to request LIBEXEC? Enter a "CAT /APP", then export the name from the list displayed on the screen by CAT.

PRUN Facility

PRO-NTO provides you with a keystroke invocation of applications via the menu displayed when PRO-NTO is activated. You can also invoke applications directly from disk via the UNIVERSAL function assigned to <F3>. The one drawback with these methods is that they require physical keystrokes detected by single key requests; thus, you cannot invoke an application from Job Control Language (JCL) using the PRO-NTO activation method as JCL requires keyboard line input.

The PRUN facility has been provided so that an application can be invoked without activation of PRO-NTO. PRUN also makes use of the KEYIN facility provided by the DOS so that it will be equally effective from JCL as well as physical keystrokes. To invoke an application using PRUN, the following example syntax may be used:

```
PRUN BRINGUP
```

which will invoke the BRINGUP application. Note that the use of PRUN still requires that PRO-NTO be installed.

A prime use for PRUN is to automatically gain access to an application when you BOOT your system. For example, if the following PRONTO/JCL file was invoked at BOOT time via an "AUTO DO PRONTO" command, when you BOOT your system disk, PRO-NTO will be installed and the BRINGUP application will be invoked:

```
PRONTO
PRUN BRINGUP
```

Invoking PSORT

The PSORT utility provides a facility for sorting data files associated with PRO-NTO applications which are designed to be ascending list ordered. PSORT is easily invoked via the command:

PSORT [file-specification [Pack]]

The "file-specification" identifies the data file which you wish to sort. The option, "Pack", is entered when you want PSORT to physically remove all "deleted" records from the data file. The first letter of "Pack" is shown capitalized. This indicates that you can abbreviate the option entry with the single letter, "P". Note that both "file-specification" and the "Pack" option are optional; however, if you want to enter the "Pack" option on the command line, you must enter the "file-specification" as well.

If you do not enter the specification on the command line, PSORT will prompt you for the information. The "file-specification" is requested via the prompt message:

Enter name of file you wish to sort >

The complete file specification must be supplied. This includes the file name, file extension, and drive specification. Next, you will have the opportunity of selecting the pack option by responding to the prompt:

Do you wish to pack the file <Y,N> ?

Respond with "Y" and any "deleted" records will be removed from the file after it is sorted.

If PSORT cannot open the file which you have identified, it will respond with the error message:

Can't open filespec

and abort the request.

Data files associated with PRO-NTO applications have information stored in the first sector of the file. This information specifies such things as the number of records in the file, the logical record length of a record, the position of the key field within the record, the length of the key field used for ordering the sort, and the first record in the file which contains data. This information is used by PSORT to control the sorting operation. PSORT will first read this information from the first sector of the file then check the validity of this information. If it detects an error when reading this data, it will respond with the error message,

Error in reading filespec

If the number of records appears out of a reasonable range, PSORT will advise

you via the message:

Can't sort ddd records

and abort the request. If the logical record length appears invalid, PSORT will advise you of this via the message:

File has invalid logical record length

and abort the request. Since the sort key must be contained entirely within the record, PSORT will make sure that based on the key's length and its relative starting position within the record that it doesn't extend past the end of the record. If it finds this condition, it will display the message:

Sort key extends past end of record

When all of the validity checks have been passed, PSORT will display the statistics of the file [number of records to sort, logical record length, key location within a record, and key length] via the informative message:

Statistics: Nrec=ddd, Lrl=ddd, Keyloc=ddd, Keylen=dd

then proceed to sort the file.

Sorting is broken up into three phases [plus a fourth phase of packing when the pack option is requested]. In the first phase, the key fields of each record are read into memory storage. In the second phase, the keys are placed into ascending alphabetic order. In the third phase, the data file is physically rearranged to match the sorted sequence of the key fields. This three phase process places certain requirements on available memory. There needs to be free memory for holding the key fields of all records as well as an index array to those fields. There also must be memory available for two file buffers. If sufficient memory is not available, PSORT will advise you of this unfortunate situation via the error message:

Can't allocate storage for keys

and abort the request to sort the data file. The solution at your disposal is to make more high memory available. If you invoke the DOS MEMORY command, you will note the value of the high memory pointer shown as HIGH$. If this is not X'FFFF', then you have modules loaded into high memory. Probably, PRO-NTO is loaded. Most likely, you have other modules loaded. You may have to reboot your system to free up the high memory then retry PSORT. Although PSORT sorts the data file directly from the disk, the key fields must fit into memory.

Assuming all goes well at this point, PSORT will advise you of the status during each phase via display of the informative messages:

Reading key fields
Sorting key fields
Rearranging data file

If you requested the "Pack" option, a fourth message,

Packing data file

will appear after the file has been sorted informing you that the packing operation has begun. The pack operation reads the file in reverse and stops when it reaches an active record. Remember that the sorting operation will push all deleted records to the end of the file; thus, reading the file in reverse reads through the deleted records. The number of records (NREC) field in the data file's information sector is also updated to reduce NREC by the number of deleted records which have been removed.

If any error occurs during the file access, one of the following error messages may be displayed:

Data file read error, record ddd
Data file write error

The first error message indicates a read error when reading record "ddd" while the second indicates an error during writing. Either error will cause the operation to abort.

Data File Information Sector - Technical Data

The first sector of any data file which can be sorted by PSORT contains information according to the PRO-NTO general data file format. Relative bytes 0-2 contain the number of data records contained in the file (0=HSB, 1=MSB, 2=LSB). Although this field has a capacity to indicate over 16 million records, PRO-NTO applications certainly are not expected to support more than a few thousand records. Other information is used by PSORT to identify the parameters needed to sort the file.

The logical record length tells PSORT how big a record is so that the file can be read properly.

The first record treated as data is used so that PSORT can ignore any portion of the file's front end [including the information sector] used to contain data which is not part of the data records. This entry is relative to record zero and is based on the file organized entirely by records of size equal to the logical record length of a record. For example, if LRL is 128 and sector 1 is the first data sector [the information sector is sector 0], FSTDAT must be set to 2. If the LRL is 512 and no additional front end data is used by the application, FSTDAT is 1 and the data starts at sector 2; thus, sector 1 remains unused.

The relative location within each record used as the sort key tells PSORT where the key is located within the record.

The length of the key is used to help establish memory requirements (along with NREC) and what fields of the record are to be compared for the sorting operation.

The following table summarizes this information. The "names" in square brackets indicate an abbreviated designation for the information items within the information sector of a PRO-NTO data file. The relative byte contents are stored in standard low-order high-order format and are indicated in hexadecimal format.

Relative	Contents
00H-02H	Number of records in the file [NREC]
10H-11H	Logical Record Length [LRL]
12H-13H	1st record of LRL treated as data [FSTDAT]
14H-15H	Relative location within each record used as the SORT key [KEYLOC]
16H	Length of the SORT key [KEYLEN]

If you are planning to write your own applications which use data files, you may want to design the information sector into your application's data file(s) so that they can be sorted by PSORT.

HELP Facility

The HELPP facility provides you with brief screens of helpful information whcih can act as a memory jogger. The HELPP facility does not replace a thorough reading of the PRO-NTO user manual. It can, however, save you from rummaging through the manual in order to locate a particular facet of PRO-NTO's operation.

HELPP is a Partitioned Data Set (PaDS) file created with the PRO-HELP package available from MISOSYS. HELPP must be invoked from DOS Ready; however, it can also be invoked via the "LIBEXEC RUN" operation available from the PRO-NTO activation menu.

If you invoke HELPP by itself, it will provide a list of names covering the topics for which help is available. This will look something like the following:

```
run helpp:3

HELP( available on:
address  bringup  calendar afpcalc  rpncalc  cardfile charset  defaults
dialer   dosave   export   import   libexec  pronto   prun     psort
svc124   term     typer    universa window   winlink
```

The actual list of names should be more extensive than what you see printed here.

The information covering a topic is displayed by entering "HELPP(" followed by the topic name. For example, if you want help on EXPORT, enter,

```
HELPP(EXPORT
```

The topic can be abbreviated to as few characters as will differentiate it from all of the other topics. In the example under discussion, the EXPORT topic may be abbreviated to "EX" giving, "HELPP(EX".

ADDRESS Data Base Application

The ADDRESS application will maintain a data file of records containing information useful for keeping a mailing list or Rotating Index File. The data file is in a format acceptable as an "ADDER" file in the PowerMAIL PLUS Data Management and Mailing List Information System. The data items available within each ADDRESS record are identified in the following table:

Field Identifier	Field Label	Width	Permissable Entries
0	Last	15	20H-7FH
1	First	10	20H-7FH
2	Company	20	20H-7FH
3	Address1	20	20H-7FH
4	Address2	10	20H-7FH
5	City	15	20H-7FH
6	State	8	20H-7FH
7	ZIP	10	20H-7FH
8	Data1	5	20H-7FH
9	Data2	12	20H-7FH
$	Flags [x]	24	20H, 'A'-'X'

Fields 0-9 are pretty straightforward. They can be used to store the typical items found in any type of address list. The DATA2 field can be used for a telephone number. In fact, by storing a phone number in this field, you can export the number to the DIAL application for automatically dialing it when you have a supported modem connected. The flag field contains 24 positions labeled 'A' through 'X' which are in either an OFF state (signified by a space) or an ON state (signified by its label character). The flags may be used to enter binary type events (YES/NO, ON/OFF). Don't forget that the flags can be the same indicators as that used in PowerMAIL PLUS (ADDRESS flags A-X correspond to PowerMAIL flags 1-24). ADDRESS makes use of the flags in both the Rotory Index CARD and MAILing label functions which can limit their output to records tagged with a flag label as one of the available options.

ADDRESS has a few commands that display a prompt message and expect a response. The response must be terminated via a depression of the <ENTER> key. During the input of the response, you can backspace over a mistyped character via the <LEFT ARROW> key. You can clear the entire response via <SHIFT> <LEFT ARROW>. You can cancel the request via the <BREAK> key. Finally, if you wish to IMPORT the response from the display screen present prior to the invocation of ADDRESS, you only need to depress the <CLEAR> <LEFT ARROW> keys. Import is fully described in the chapter on PRO-NTO operation.

When you invoke the ADDRESS application, it will request PRO-NTO to open a window. If a window cannot be opened, a short beep will sound from your computer's internal speaker and ADDRESS will terminate. This happens when you have exceeded the maximum number of windows that can be open at one time.

That's a rare instance. When the window is opened, ADDRESS will search all disk drives for a file named ADDRESS/DAT. If you want to maintain more than one discrete address list file, just keep each one on a separate floppy disk. When an ADDRESS/DAT file is found, it will be accessed and the first address record, if any, will be displayed. If no file can be found identified as ADDRESS/DAT, you will be prompted to specify the drive which should be used to create an ADDRESS/DAT file via the prompt:

Create ADDRESS/DAT on what drive?

Respond with one of your available drives (0-7). Once the ADDRESS/DAT file is created, the displayed window will show the record fields and ADDRESS commands which are available. A filled in record appears as follows:

```
+-----------------------------------------------------------+
| Last :Soltoff        : First :Roy       :    Data1 :85-04: |
| Company :MISOSYS, Inc        :        Data2 :703-450-4181: |
| Address1 :PO Box 239          :     Address2 :          : |
| City :Sterling       : State :VA        : Zip :22170-0239: |
|            Flags [ ] :                  RS      :          |
|-----------------------------------------------------------|
| This line is used to display messages!                    |
| ==> Add Card Delete Edit First Last Mail Next Prev Search  |
+-----------------------------------------------------------+
```

When ADDRESS is waiting for a command entry, the arrowhead which points to the list of available commands will be blinking. These commands permit a range of operations on the data records. A command is invoked by depressing the letter key noted by the first letter of the command (the letter which appears capitalized). This may be entered in either upper or lower case. You can also invoke EXPORT of data when ADDRESS is waiting for a command. Data export permits you to pass a specified field or block of data from the displayed window to the environment interrupted when you invoked ADDRESS. More information on EXPORT can be found in the chapter on PRO-NTO operation. Details for each command follow:

ADD a record

When you wish to add a record to your address file, specify this action by depressing the <A> key. You will be presented with a display where the contents of each field are blank. ADDRESS will then automatically enter the EDIT command. Each field can be edited with the entry of data (or editing within a field if you mistype). If you decide that you do not want to add the record, just depress <BREAK>. ADDRESS will abort the addition and display whatever record was on the window at the time that ADD was invoked.

Print a Rotory Index CARD

This command is used to print the contents of a record or the entire data file in a format suitable for 4" wide Rotory Index Cards in continuous forms. You can even specify that the output is to be limited just to records which have a particular flag turned ON (i.e. one of the flags A-X). When you invoke CARD, you will be prompted to select the output option via the prompt:

<A>ll, >C<urrent, or <Fx>lagged?

If you want the entire data file printed, enter an <A>. If you want the output to be limited to flagged records, enter an <F> followed by the flag letter for which you want to limit the output. For instance, if you want to print CARDs for all records with flag-P set, enter the two letters "FP" followed by <ENTER>. If you just want the current record printed on the "card", enter the letter <C>. If you respond with no entry (by depressing just <ENTER>), the default of CURRENT will be selected. This default is indicated by the reversed angle brackets surrounding the letter "C" in the prompt. If you specify ALL records or FLAGGED records, any that are stored with a "deleted" mark will not be selected for printing.

You will then be asked to select the printing format via the query:

Format? >

If you just depress the <ENTER> key, you will select the default format supplied with ADDRESS. This format is:

```
|
|
|LAST---------- FIRST-----  DATA2
|COMPANY-------------   DATA1-------
|ADDRESS1------------  ADDRESS2--
|CITY----------- STATE--- ZIP-------
|FLAGS--------------------
|
|
|
|
|
|
```

The vertical lines are not part of the output but are only illustrated here to denote the beginning of a line. Note that there are 13 lines total. This format is tailored to 13 lines per card - the quantity of lines on a Rotory Index Card at six lines per inch. Each line of data will also fit on a card printed at 10 characters per inch.

The format is printed by ADDRESS according to a string of field and printing specifiers. The string corresponding to the default format is:

```
';;0 1  9;2   8;3  4;5 6 7;$;;;;;;;',CR
```

The contents of a particular field are printed when a string character is one of the field identifiers 0, 1, 2, 3, 4, 5, 6, 7, 8, or 9. The flag field is specified for printing via the dollar sign character, "$". This field will be printed with a SPACE for each flag that is OFF and the flag label for each flag that is ON. A semicolon is interpreted as a logical carriage return. Any other character is interpreted as a character to print. A carriage return terminates the string without taking any printing action. If you want to specify some other format, just enter a string of characters in response to the "Format?" prompt. A string of up to 48 characters may be entered. If you are interested in maintaining your own formats, why not keep the "strings" in a record of the CARD application? By doing that, you can IMPORT the string when prompted for "Format ?".

If you wish to abort printing at any time, just depress the <BREAK> key. When <BREAK> is detected, ADDRESS will output an immediate carriage return and return to await a command. The window will display the record being output at the time the <BREAK> was detected.

If the printer is not available during the printing operation, ADDRESS will display the error message,

Device not available

and terminate the CARD command. ADDRESS will return to await a new command.

DELETE currently displayed record

This command is used to delete the record currently being displayed. You have to confirm your request by responding to the query:

Delete: <Y> to confirm?

Any response other than the letter <Y> will abort the delete operation and return to the command prompt to await a new command. If you confirm the deletion, it is marked in the following manner. The contents of the LAST name field are shifted right by one position with the 15th character being dropped. A character value of 255D is placed in the first character position and the record is filed. This indicates a "deleted mark". A deleted record is still displayed if you come to it while "scrolling" through the data base; however, it will not be printed during a "CARD" or "MAIL" operation when ALL or FLAGGED records are requested nor will it be matched during a SEARCH. If you examine position 255 in the CHARSET application display, you will be able to note the "deleted" mark character displayed by your computer. Deleted records can be permanently removed from the data file by requesting the "PACK" option during a sort of the data file via the PSORT utility provided

with PRO-NTO.

It is perfectly acceptable to "un-delete" a deleted record by editing the LAST field and deleting the "deleted mark". It is not possible to restore a deleted record after a PSORT operation where you have requested the "PACK" option - at least not with any utility provided with PRO-NTO!!!

EDIT the currently displayed record

EDIT is selected to enable you to change any of the data in any field. When EDIT is invoked, the following message is displayed in the bottom portion of the window:

Edit: ^F to file; BREAK to abort

This is an aid to letting you know that there are two ways of escaping from the edit mode. One way is to enter <CONTROL-F> to file away your edited record. The other way is the depress the <BREAK> key which will abort the editing and return to await a new command. Also, a blinking cursor will appear in the first character position of field zero (LAST). When the cursor is first moved to a field, it can be moved to the next higher field identifier via the <RIGHT ARROW> key or the <DOWN ARROW> key. The cursor can be moved to the next lower field identifier via the <LEFT ARROW> key or the <UP ARROW> key. The FLAG field is considered field ten (10). When advancing towards lower numbered fields, field zero (LAST) wraps to field ten (FLAGS). When advancing towards higher numbered fields, ten wraps to zero.

When you are positioned at the field you want to edit, you can start editing it by one of two methods. If the first character is to be changed, just overstrike it with the correct character; this also begins the field edit mode. If some other character is to be changed, depress the <ENTER> key to enter the field edit mode; thereafter, the <LEFT ARROW> and <RIGHT ARROW> keys will position the cursor left and right respectively within the field.

You can insert a character by moving the cursor at the insertion position and depressing CONTROL-A (for "add" of a character). The characters extending from that position through to the end of the field will be shifted right and a space will be inserted. To delete a character, depress CONTROL-D (for "delete" a character). The character at the cursor position will be deleted and all trailing characters in the field will be shifted left to "take up the slack". When you are finished editing a field, depress the <ENTER> key and the cursor will be advanced to the next higher numbered field. Again, field ten will wrap to field zero.

The flag field is a special case. When the cursor is placed in this field, the flag label corresponding to the cursor position is identified within the square brackets of the field label. This label is a letter in the range 'A', 'B', 'C', ... 'V', 'W', 'X'. A BLANK is displayed anytime the cursor is positioned within one of the other fields 0-9. The entry characters usable are the SPACE which will denote an OFF state and any other printable

character which indicates an ON state. When you turn ON a flag, the appropriate flag label will be displayed regardless of what character you enter to turn ON the flag. The <LEFT ARROW> and <RIGHT ARROW> keys will position you left and right accordingly once you enter the field edit mode either by depressing <ENTER> or a character.

When you have completed your edits, depress <CONTROL-F> to file them away. Don't forget that if you abort the editing by depressing the <BREAK> key, the record will be left as it was prior to invoking EDIT.

Position to FIRST record

This command is used to rapidly position to the first data record of the ADDRESS/DAT data file and display it.

Position to LAST record

This command is used to rapidly position to the last data record of the ADDRESS/DAT data file and display it.

Print a MAILing label

This command is used to print the contents of a record or the entire data file in a format for one-across mailing labels on continuous forms. You can even specify that the output is to be limited just to records which have a particular flag turned ON (i.e. one of the flags A-X). When you invoke MAIL, you will be prompted to select the output option via the prompt:

<A>ll, >C<urrent, or <Fx>lagged?

If you want the entire data file printed, enter an <A>. If you want the output to be limited to flagged records, enter an <F> followed by the flag letter for which you want to limit the output. For instance, if you want to print mailing labels for all records with flag-P set, enter the two letters "FP" followed by <ENTER>. If you just want the current record printed on the label, enter the letter <C>. If you respond with no entry (by depressing just <ENTER>), the default of CURRENT will be selected. This default is indicated by the reversed angle brackets surrounding the letter "C" in the prompt. If you specify ALL records or FLAGGED records, any that are stored with a "deleted" mark will not be selected for printing.

You will then be asked to select the printing format via the query:

Format? >

If you just depress the <ENTER> key, you will select the default format supplied with ADDRESS. This format is:

```
|FIRST----- LAST-----------
|COMPANY-------------
|ADDRESS1------------  ADDRESS2--
|CITY----------- STATE--- ZIP-------
|
|
```

The vertical lines are not part of the output but are only illustrated here to denote the beginning of a line. Note that there are six lines total. This format is tailored to six lines per label - the quantity of lines on a 15/16" high label at six lines per inch. Each line of data will also fit on a 3-1/2" wide label printed at 10 characters per inch.

The format is printed by ADDRESS according to a string of field and printing specifiers. The string corresponding to the default format is:

```
'1 0;2;3 4;5 6 7;;;',CR
```

The contents of a particular field are printed when a string character is one of the field identifiers 0, 1, 2, 3, 4, 5, 6, 7, 8, or 9. The flag field is specified for printing via the dollar sign character, "$". This field will be printed with a SPACE for each flag that is OFF and the flag label for each flag that is ON. A semicolon is interpreted as a logical carriage return. Any other character is printed. A carriage return terminates the string without taking any printing action. If you want to specify some other format, just enter a string of characters in response to the "Format?" prompt. A string of up to 48 characters may be entered. If you are interested in maintaining your own formats, why not keep the "strings" in a record of the CARD application? By doing that, you can IMPORT the string when prompted for "Format ?".

If you wish to abort printing at any time, just depress the <BREAK> key. When <BREAK> is detected, ADDRESS will output an immediate carriage return and return to await a command. The window will display the record being output at the time the <BREAK> was detected.

If the printer is not available during the printing operation, ADDRESS will display the error message,

Device not available

and terminate the MAIL command. ADDRESS will return to await a new command.

Position to NEXT record

This command is used to display the next data record in your address data file. You can also advance to the NEXT record by depressing the <DOWN ARROW> key.

Position to PREVIOUS record

This command is used to display the previous data record in your address data file. You can also position to the previous record by depressing the <UP ARROW> key.

SEARCH for a particular record

The SEARCH command is used to scan the data file starting at the record which follows the current record. The search string is entered in response to the query:

Search for? >

You can enter up to a 24-character string. If you enter a NULL string (i.e. <ENTER> by itself), the previous search string will be reused. The search string is not disturbed by intervening non-SEARCH commands. In this way, repetitive search commands can be invoked to display all records which match a single search string. The search string will be matched against each record until a match is found. The LAST and FIRST fields of the record will be considered as a single 25-character key field. If your search string is a sub-string of the record's key, the search will stop and that record will be displayed. The search will also stop at the last record. If a record cannot be found which matches up with your search string, the message,

Record not found

will be displayed and the current record will remain in the window.

Sorting your ADDRESS/DAT file

You can sort your data file into ascending order according to the key string composed of the LAST and FIRST fields by invoking the PSORT utility from DOS Ready. This is done via the command:

PSORT ADDRESS/DAT [pack]

\fr tsehtgiy sipi;rerpV si v Vhc rgerpr' lfrg at' lhcf et pr;t:r efr 'r leted records from your address file. The square brackets are not entered.

Disk errors

If, by chance, ADDRESS detects an error in reading or writing your address file's records, it will display the DOS error message in the message line at the bottom of the window and then await an entry from the keyboard. This gives you an opportunity to read the error message. After you depress any key, ADDRESS will terminate its operation and return you to the program that was running when you invoked ADDRESS.

ADDRESS/DAT Technical Specifications

Each record is stored in 128 bytes starting with the second sector of the ADDRESS/DAT data file. The record is composed as follows:

Field Identifier	Field Label	Width	Record Location
0	Last	15	000-014
1	First	10	015-024
2	Company	20	025-044
3	Address1	20	045-064
4	Address2	10	065-074
5	City	15	075-089
6	State	8	090-097
7	ZIP	10	098-107
8	Data1	5	108-112
9	Data2	12	113-124
$	Flag [xx]	3	125-127

The flags are stored in a 3-byte field - each flag occupying one bit. The low-order bit representing a lower flag letter. Thus, byte 125 bit-0 is flag A while byte 127 bit-7 is flag X.

The first sector of the file contains information according to the PRO-NTO general data file format. Relative bytes 0-2 contain the number of data records contained in the file (0=HSB, 1=MSB, 2=LSB). This conforms to the PowerMAIL PLUS format for an ADDER file (PMAIL/ADD). Additional information used by PRO-NTO is as follows with the numbers in square brackets indicating the quantities fixed within the ADDRESS/DAT data file (relative byte contents are stored in standard low-order high-order format):

Relative	Contents
10H-11H	Logical Record Length [128D]
12H-13H	1st record of LRL treated as data [02D]
14H-15H	Relative location within each record used as the SORT key [0D]
16H	Length of the SORT key [25D]

BRINGUP Tickler File and Appointment Book

BRINGUP maintains a data file of activities to be displayed by date in the application's window. This is sometimes called a "tickler" file. The activities may indicate things that should be done on a particular date or appointments that must be kept at a particular time. The only limit to the number of total activities that can be stored is the allowable disk file size on your disk; however, only a maximum of twelve activities may be stored for any one date. BRINGUP supports the same range of dates as that supported by the DOS [01/01/80 through 12/31/87]. Each data activity record contains the following items:

Field	Description of Intended Use
Checkoff	Used to flag the activity as completed;
Priority	Used to denote an activity priority <1-4>;
Time	Used to denote the time of the item in hours and quarters of an hour;
Text	Describes the record's activity (29 chars)

When you invoke the BRINGUP application, it will request PRO-NTO to open a window. If a window cannot be opened, a short beep will sound from your computer's internal speaker and BRINGUP will terminate. This happens when you have exceeded the maximum number of windows that can be open at one time which is a very rare instance. When the window is opened, BRINGUP checks for a file named, BRINGUP/DAT. If it is found, BRINGUP will use the records in that file. If one is not found, BRINGUP will prompt you for the disk drive number to create the data file via the prompt:

Create BRINGUP/DAT file on what drive?

Respond with one of your available drives (0-7). Once the BRINNGUP/DAT file is created, BRINGUP will establish the window with "today's" records - "today" being the current system date set at powerup or via the DATE library command. An alternative method of invoking BRINGUP is from the CALendar application. You can use CALendar's pointer to establish a BRINGUP date which will be passed to BRINGUP if invoked from CALendar. This date then becomes the "current" date rather than today's date. The screen will always show the current date in the lower right portion of the display. BRINGUP will turn on the time-clock feature of the DOS. The time clock will normally be displayed in the top window border as illustrated in the window facimile below. The time will be displayed during the duration of BRINGUP's operation. If you nest to another application from BRINGUP, the time-clock will remain visible until you exit from BRINGUP [note: if the clock was on when BRINGUP was invoked, it will remain on when you exit BRINGUP]. A blinking cursor pointer, ">", will be positioned on the screen pointing to "the current activity". This may be positioned up or down via the <UP ARROW> or <DOWN ARROW> keys.

The format of an activity's entry is PRIORITY, followed by TIME, followed by the text which describes the activity. If the activity has been checked off, a check mark (machine dependent) will appear prior to the pri-

ority field. All of the activities for the date displayed will appear in sequential order sorted by time (using a 24-hour clock). BRINGUP does not restrict more than one activity from having the same activity time. In fact, where you have activities that are date sensitive but not time sensitive, you may want to assign such activities to an arbitrary time slot.

```
+--------------------------------HH:MM:SS--+
|>   1 12:00 Jeep payment due               |
|    1 12:00 Pay phone bill                 |
|    1 12:00 Pay Exxon bill                 |
|    2 13:45 Meet with accountant           |
|    3 16:30 Schedule meeting with Karl     |
|                                           |
|                                           |
|                                           |
|                                           |
|                                           |
|                                           |
|                                           |
|-----------------------Wed, Mar 13, 1985--|
| This line is used to display messages!    |
|Add Chk Del Echo File Goto Mov Nxt Prv Rmv|
+-------------------------------------------+
```

Anytime a prompt does not appear in the status line, BRINGUP is awaiting a command. The commands available to you are displayed at the bottom of the window. These commands are:

Command	Purpose of Command
ADD	Used to add a record to "today".
CHECK	Used to check off the "current" activity.
DELETE	Used to delete the current activity.
ECHO	Used to copy the current activity to another date. Useful to propogate a monthly activity.
FILE	Used to file activity changes.
GOTO	Used to switch to a different date.
MOVE	Used to move the current activity to another date.
NEXT	Used to advance to the next consecutive date.
PREV	Used to step back to "yesterday".
REMOVE	Used to purge activities from a range of dates.

In addition, <BREAK> will exit the application; <CLEAR> <RIGHT ARROW> will invoke EXPORT of data while <CLEAR> <LEFT ARROW> invokes import of data. The functions of both IMPORT and EXPORT are fully described in the chapter on PRO-NTO operation.

ADD Activity

ADD is the means by which activities are added to your bringup data base. ADD will prompt you first for the priority via the prompt:

Priority <1-4> ?

This may be used by you to indicate how "hot" is the item. It could also be used to note which of four individuals should complete the activity - if you hapen to be the boss and delegate certain tasks. The priority is a number from 1 to 4. If you enter any character other than a number in the range 1-4, you will be re-prompted for the entry. A <BREAK> will abort the addition.

Next you will be asked to enter the time of the activity. You may enter an hour and quarter of an hour in response to the prompt:

Time <HH:MM> ?

The "HH" field accepts hours from 00 through 23 (two digits are required). The "MM" field accepts minutes from 00 through 59 (two digits are required) but will truncate downward to the four values: 00, 15, 30, 45 (quarters of an hour). If your entry does not adhere to the syntax illustrated, you will be re-prompted for a correct entry. As in the priority response, a <BREAK> will abort the addition.

Next you will be asked to enter the text. You can enter a string of up to 29 characters to describe the activity. As before, if you depress <BREAK> in response to the text query, it will abort the addition.

When you complete the addition, the message, "Changes to file", will be displayed to the left of the current date if it hasn't already been displayed from an earlier addition. This message indicates that the addition will not be stored until a FILE command is invoked. Thus if you exit from BRINGUP without filing your changes [you will be prompted if there are changes to be filed], then the additions will not become permanent. Please note, however, that if you MOVE or ECHO to another date, any changes made in the current date will be automatically filed.

CHECK off an Activity

This command will place a check mark on the screen preceding the priority field of the current activity [note: The check mark is displayed via character value 236 (decimal) - See CHARSET for your computer's display]. The message, "Changes to file", will be displayed to the left of the current date. This indicates that the checkoff will not be stored until a FILE command is invoked. If you want to "uncheck" an activity, just "CHECK" it again. The CHECK command toggles the state of the "completed" field.

DELETE an Activity

This command will delete the current activity and remove it from the window screen. The message, "Changes to file", will be displayed to the left of the current date. This indicates that the deletion will not be stored until a FILE command is invoked. If you want to delete a large number of activities, i.e. all activities of a particular date or range of dates, you probably want to use the REMOVE command available in BRINGUP.

ECHO Activity to Another Date

This command will copy the current activity to another date without deleting it from the current date. This type of command is useful when you want to propogate a monthly activity across a number of months. Enter the date you wish to copy the activity to in response to the prompt:

Move to date <MM/DD/YY> ?

Note that although the query says "Move to", you will be copying the activity. The system date will automatically be used if you do not enter a date but just depress <ENTER> in response to the query. In this way, once you change the current date from "today", you can easily copy an activity to "today" via "ECHO ... <ENTER>". BRINGUP will save the activity's text field. The activity will not be deleted, changes will be filed, and the echo date will become the current date. The window screen will display any records for the new current date and then go automatically into the ADD command. If, by chance, the echo date already has twelve activities stored, you will not be able to add the moved text. The text will still be stored so that you could GOTO another date. When ADD takes control, it recognizes the saved text and will prompt you for only the PRIORITY and TIME.

FILE Changes

This command will update your bringup file with all changes made to the current date. The message, "Changes to file", will be removed.

GOTO Another Date

This command is used to change the current date to some other date. If there are any changes in the "old" current date, you will be flagged prior to the GOTO via the query:

File your changes <Y,N> ?

If you respond with <Y>, the GOTO will be canceled and changes will be filed. An <N> response will continue the GOTO operation without filing any changes for the current date.

You will be asked to enter a new date via the prompt:

Goto date <MM/DD/YY> ?

The date is entered according to the format illustrated in the prompt. Your entered date will become the new "current" date. If you do not enter a date but just depress <ENTER>, the system date will automatically be used. In this way, once you change the current date from "today", you can easily return to "today" via "GOTO <ENTER>". The data file will then be searched to display the activities for the requested date.

MOVE Activity to Another Date

This command will move the current activity to another date. Enter the date you wish to move the activity to in response to the prompt:

Move to date <MM/DD/YY> ?

The system date will automatically be used if you do not enter a date but just depress <ENTER>. In this way, once you change the current date from "today", you can easily move an activity to "today" via "MOVE <ENTER>". BRINGUP will save the activity's text field. The activity will be deleted, changes will be filed, and the move date will become the current date. The window screen will display any records for the new current date and then go automatically into the ADD command. If, by chance, the move date already has twelve activities stored, you will not be able to add the moved text. The text will still be stored so that you could GOTO another date. When ADD takes control, it recognizes the saved text and will prompt you for only the PRIORITY and TIME.

Go to NEXT date

The GOTO command allows you to advance rapidly to any designated date within the range of dates supported by BRINGUP. When you want to look at the activities scheduled for "tomorrow" or the next day after tomorrow, use the NEXT command. It advances the date by one day. The end of the month advances to the first of the next month. The end of the year advances to the first of January of the next year. The last day of the supported dates (December 31, 1987) advances to the first day of the supported dates (January 1, 1980). Each time you go to the NEXT day, the data file will be searched for all of the activities scheduled for that date. Any that are "current" will be sorted and displayed on the window.

Go to PREVIOUS date

The GOTO command allows you to advance rapidly to any designated date within the range of dates supported by BRINGUP. When you want to look at the activities scheduled for "yesterday" or the day before yesterday, use the

PREV command. It decrements the date by one day. The first of the month decrements to the last day of the previous month. The first day of the year decrements to December 31st of the previous year. The first day of the supported dates (January 1, 1980) decrements to the last day of the supported dates (December 31, 1987). Each time you go to the PREV day, the data file will be searched for all of the activities scheduled for that date. Any that are "current" will be sorted and displayed on the window.

REMOVE Activities

This command can be used to remove all activities from a range of dates. Normally, activities are removed one at a time via the DELETE command. When you go to a date, the entire data file is searched for activities which "belong" to that date. Thus, if the data file is encumbered by many old activities (obviously which have been completed and checked off), the search time may grow excessive. Therefore, you may find it convenient to purge very old activities from your data file so that the space they occupy can be re-used by newly added activities. REMOVE is used for this purpose. The REMOVE command will prompt you to enter the date range via the prompt:

Remove range ?

This may be answered according to the following syntax table:

Date Syntax	Interpretation of Range
M1/D1/Y1-M2/D2/Y2	Remove from date1 through date2 inclusive.
-MM/DD/YY	Remove from 01/01/80 through date inclusive.
MM/DD/YY-	Remove from date through 12/31/87 inclusive.
MM/DD/YY	Remove activities only on date.

Only those activities within the date range you specify that have been checked off will be removed. At the conclusion of the removing operation, the window screen will be returned to "today's" date.

BRINGUP/DAT Technical Specifications

For the technically inclined, the following describes the makeup of activities stored in the data file. It is quite possible to access the file and its records from other programs. Each activity is stored in a 32-byte record. Thus, each sector of the data file can contain eight activities. The record is "fielded" as follows:

Byte 0:
- bits 7-6 - Priority minus one [0-3].
- bit 5 - Set to a one if the activity is "checked".
- bit 4 - Set to a one if the record is ACTIVE. A zero implies that the record has been DELETED or PURGED.
- bits 3-0 - The month portion of the date [1-12].

Byte 1:
   bits 7-3 - The day portion of the date [1-31].
   bits 2-0 - The year portion of the date excess 1980 [0-7].

Byte 2:
   bit   7 - Currently unused.
   bits 6-2 - The hour portion of the time [0-23].
   bits 1-0 - The quarter-hour portion of the time [0-3].

Bytes 3-31: The text of the activity.

The following simple BASIC program [no frills, folks] has been provided to permit you to externally display all of the records in your BRINGUP/DAT file.

```
10 REM bringup/bas - print the bringup/dat file
20 OPEN "D",1,"BRINGUP/DAT",32
30 FIELD 1, 1 AS B1$, 1 AS B2$, 1 AS B3$, 29 AS M$
35 U$="& # ##/##/## ##:## &"
40 IF EOF(1) THEN CLOSE 1:END
50 GET 1
60 B1% = ASC(B1$)
70 IF (B1% AND &H10) = 0 THEN GOTO 40
75 B2% = ASC(B2$): B3% = ASC(B3$)
80 PR% = ((B1% AND &HC0) / 64) + 1
90 IF (B1% AND &H20) = 0 THEN C$=" " ELSE C$="c"
100 MM% = B1% AND &HF
110 YY% = (B2% AND &H7) + 80
120 DD% = (B2% AND &HF8) / 8
130 HH% = (B3% AND &H7C) / 4
140 QQ% = (B3% AND 3) * 15
150 PRINT USING U$;C$,PR%,MM%,DD%,YY%,HH%,QQ%,M$
160 GOTO 40
```

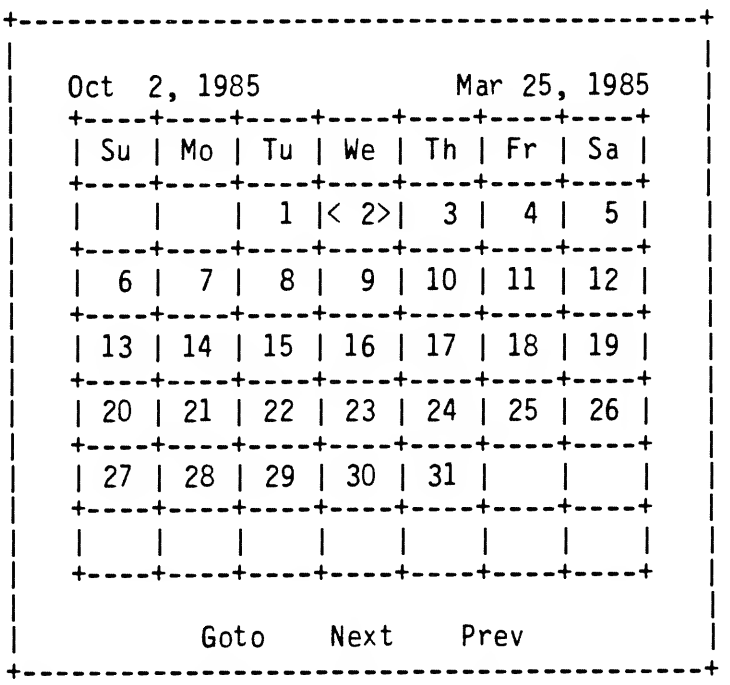
You can easily change the one PRINT statement to an LPRINT statement in order to print the items on a printer. The program displays all active records and unpacks the data for printing. Records which have been checked off are prefixed with a "c". A sample display output follows.

```
  1  4/13/85 12: 0 Jeep payment due
  1  4/ 2/85 10: 0 GHA appt for Stefanie
c 1  3/27/85 12: 0 PB meter rental bill
  1  4/15/85 11:45 File IRS form 941 for 1Q85
  1  6/13/85  9: 0 VA est corp tax, payment 3
  1  9/13/85  9: 0 VA corp est tax payment 4
  1  9/13/85  9: 0 Fed 1120 Corp est tax
  1  6/13/85  9: 0 Fed 1120 Corp est tax
  2  4/15/85  9: 0 VA tax w/h due
c 1  3/22/85 12: 0 GHA payment due
  1  4/15/85 11:30 File VEC 1Q85 report
```

CALendar Application

The CALendar application can be used to obtain a one month page out of a calendar for any year you choose between 1582 and 4902 inclusive. The application provides you with commands to scroll through month after month after month. CALendar also can pass a selected date to BRINGUP in order to override BRINGUP's default initial presentation of the system date and cause it to present your selected date.

When you invoke the CALendar application, it will request PRO-NTO to open a window. If a window cannot be opened, a short beep will sound from your computer's internal speaker and CALendar will terminate. This happens when you have exceeded the maximum number of windows that can be open at one time. That's a rare instance. When the window is opened, CALendar will display the calendar page for the current month. The calendar application will display a month at a glance for just about any month you request. The displayed date can be easily changed via single key commands that adjust the display to the date of your choice. A typical calendar display appears as follows:



The current day will be marked by a flashing set of angle brackets. In the example illustrated, the 2nd day of October, 1985 is "marked". At the top left of the display, the selected day is noted as a date string. The upper right portion of the display will always display a date string which indicates today's date - today being the system date established at powerup of your computer or via the DATE command.

You can position the marker to select any date of your choosing by using the following keys:

Keystroke	Marker movement
<RIGHT ARROW>	- position to next day
<LEFT ARROW>	- position to previous day
<UP ARROW>	- position to previous week
<DOWN ARROW>	- position to next week
N	- position to next month
P	- position to previous month

You can also select a date for display via the GOTO command. If you depress <G>, you will be prompted to enter a date in response to the prompt,

Goto date <MM/DD/YY> ?

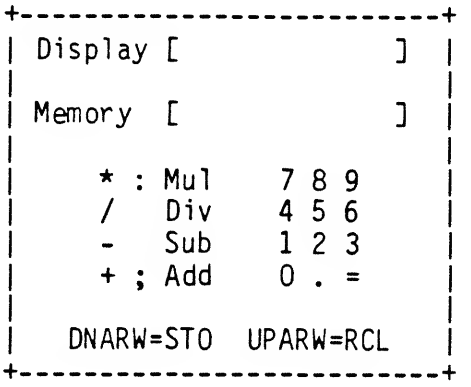
Select any date according to the entry syntax shown. The date is assumed to be in the 20th century. If you press <ENTER> without a date, it will default to the system date (i.e. today).

If you set CALendar to a particular date and then invoke BRINGUP, the current date established by BRINGUP will be the day marked by CALendar, rather than "today" - provided the date is in the range of dates acceptable to the BRINGUP application.

Algebraic Floating Point Calculator Application

The AFPCALC application provides a basic four-function (add, subtract, multiply, and divide) floating point calculator. The calculator has a standard register for making entries and displaying the results of a calculation or series of calculations. This register is labeled, "Display". The AFPCALC screen also supports a memory storage register so that you can STORE an entry or result and later RECALL it for use in additional calculations. This register is labeled, "Memory".

When you invoke the AFPCALC application, it will request PRO-NTO to open a window. If a window cannot be opened, a short beep will sound from your computer's internal speaker and AFPCALC will terminate. This happens when you have exceeded the maximum number of windows that can be open at one time. That's a rare instance. When the window is opened, you will be greeted by a display which looks like the following screen presentation:



The register labeled "Display" presents your entries and results within the area bounded by the left and right square brackets "[]". Anytime a number is displayed, it will be presented in one of two forms. Numbers in the range .1000000 through 9999999 will be displayed in decimal notation as shown. Numbers outside of that range are displayed in scientific notation (i.e. [1.234567E+10] with one digit to the left of the decimal point and six digits to the right of the decimal point. The "E" indicates the exponent which will be positive or negative according to the sign which immediately follows the "E". The exponent is a two digit base 10 number. If the value displayed is negative, a minus sign will appear in the first character position (which is shown here as blank). A positive quantity will be indicated by the absence of a minus sign.

The largest possible positive number which can be used is approximately 1.70411E+38. Any calculation which exceeds this amount will result in a flashing display. The flashing display indicates that the result is outside of the range of values acceptable to AFPCALC. The smallest possible positive number is approximately [1.162080E-38]. AFPCALC supports 6-7 digits of precision.

The "Memory" may be used to store any figure displayed in the "Display" register - even during an entry. This is done by just depressing the <DOWN ARROW> key. Anytime you wish to use the stored value in a calculation, just recall it by depressing the <UP ARROW> key. If you recall the memory while you are in the process of entering a number but have not yet completed the entry, the recalled number will replace your partial entry.

The calculator window shows the keys associated with the four functions: <*> for multiplication, </> for division, <-> for subtraction, and <+> for addition. Since the <+> key and the <*> key are shifted keys on the keyboard, AFPCALC also lets you use the unshifted keystrokes <:> and <;> respectively, for the ease of entering these two operators. The <ENTER> key is used to indicate an "equals sign" in order to obtain the result. You may also use the <=> key if you so desire.

Entries are made purely algebraicly; however, AFPCALC does not support any hierarchy of operators nor the use of parentheses. For example, the calculation:

47.36 + 39.24 * 17.22 / 4.76 =

will produce the result, [313.2883]. The calculations are performed sequentially with the intermediate results chained to the next entry. It is possible to chain an operator, such as <+>, to be able to raise a number to a power. As an example, the repeated calculation:

2 + + + + + + + + + + =

will produce the results, 2 4 8 16 32, ..., [2048.000].

You may IMPORT an entry or a series of entries. For example, the two previous illustrations were actually calculated by importing the text string from the SCRIPSIT file being edited into AFPCALC which was invoked while the AFPCALC documentation was being prepared with SCRIPSIT. The result was also EXPORTed back to SCRIPSIT and automatically input into the text file being edited. This achieved the advantage of 100% accuracy. You can do the same thing - importing a series of numbers to be calculated from some application interrupted by an AFPCALC invocation and having the result exported back to the application. AFPCALC will ignore any character that is not a valid entry character; thus, the imported "string" of values can really be free formatted.

When you are manually keying in a number, you can backspace over a mistake by depressing <LEFT ARROW>. If you wish to "clear" the display register entry , enter the two-key combination, <SHIFT> <LEFT ARROW> This will erase the value currently being displayed but it will not effect any pending intermediate result. If entered after an <=> or <ENTER>, the <SHIFT> <LEFT ARROW> combination will also clear the result so that you can start anew. The entry of a new value will also clear the result; however, if you enter an operator, the previous result will be considered to be a value entry. For

example, the chained calculation:

47.36 + 39.24 = * 17.22 = / 4.76 =

will produce the same result as before, [313.2883].

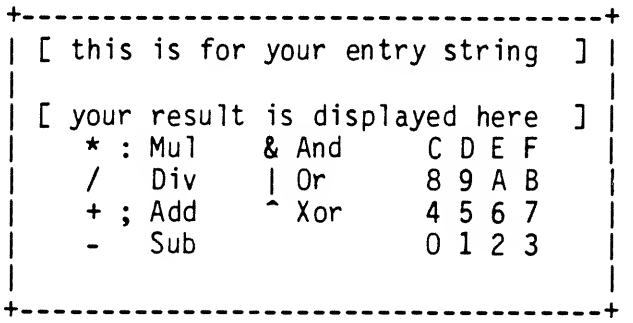
When you are entering a number, only one decimal point will be permitted. If you enter it in the wrong position, you can backspace over it and get an opportunity to enter it in a different place. Likewise, if you are entering a number in scientific notation, the "E" can only be entered once; however, you can backspace over it and re-enter it in a different position. The "E" must be followed by a plus sign, a minus sign, or a digit <0-9>. Once you enter one of the three, the next plus sign or minus sign will be considered to be an operator (addition or subtraction). You can also backspace over the exponent's sign if you enter the wrong character.

All in all, AFPCALC puts a powerful four-function calculator right at your fingertips available at the touch of a button.

RPN Calculator Application

The RPNCALC application provides a standard four-function (add, subtract, multiply, and divide) reverse polish notation 16-bit "programmer's calculator with three additional logical operators. You can calculate in decimal, in hexadecimal, in octal, or in binary. The calculator provides you with a string field for making entries and another for displaying the results of a calculation or series of calculations.

When you invoke the RPNCALC application, it will request PRO-NTO to open a window. If a window cannot be opened, a short beep will sound from your computer's internal speaker and RPNCALC will terminate. This happens when you have exceeded the maximum number of windows that can be open at one time. That's a rare instance. When the window is opened, you will be greeted by a display which looks like the following screen presentation:



The RPN calculator supports the following four types of numbers:

Syntax	Number type
bbbbB	binary
ooooO	octal
ddddD	decimal (default)
xxxxH	hexadecimal

The seven operators at your disposal are selected by entry of the following single keystrokes:

Key	Mathematical operation
*	Multiplication
/	Division
+	Addition
-	Subtraction (negation is not supported)
&	Bitwise AND
\| | Bitwise OR |
^ | Bitwise XOR |

You may enter the multiplication operator in lower case [i.e. the colon] or the addition operator in lower case [i.e. the semi-colon].

The result is obtained by entry of an equals sign "=" which must be entered with the <SHIFT> key as the unshifted keystroke is a minus sign. The entry string is terminated with an <ENTER>. The default output base will be decimal. If you wish to output the answer in any base other than decimal, follow the "=" with a "b", "o", or an "h" to specify binary, octal or hexadecimal respectively. Any numeric entry can also be entered in something other than decimal by appending the base suffix to the number string [i.e. 1a9h]. Entering a period will cause the last result to be substituted.

The calculator fully supports both EXPORT and IMPORT. Thus, your input string could be imported from some other program or application with the result being exported back to the program.

The following examples illustrate sample calculations.

Example 1:

Multiply the hexadecimal number 22 by the binary number 1111, add the decimal number 2, then output the result in octal.

22h 1111b * 2 + =o<ENTER>

The result will be shown as "001000" in the result field.

Example 2:

Output the result of example 1 but in binary:

. =b

The result will be shown as, "00000010 00000000".

CARD Filer Application

The CARD filer and notepad application maintains a data file of records - each record containing up to 480 characters of textual data. Three additional fields provide a means of identifying the text with an ID name as well as a date/time stamp as to when the card was last changed. The text is organized as 12 rows of 40 columns in width. This format is suitable for printing on continuous form 3 x 5 cards. The data items available within each CARD record are identified in the following table:

Field Identifier	Field Label	Width	Permissable Entries
1	TEXT	40 x 12	>= 20H
2	Number	0	n/a
3	ID	8	20H-7FH
4	Date	8	n/a
5	Time	8	n/a
6	reserved	8	n/a

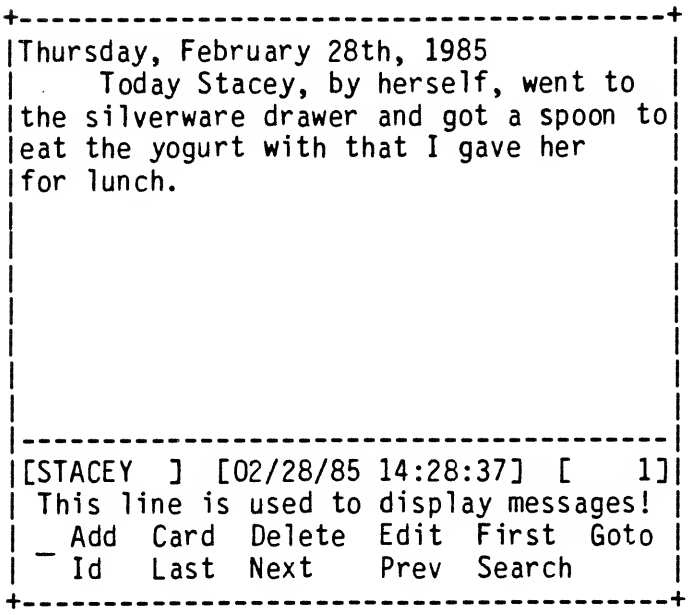
CARD has a few commands that display a prompt message and expect a response. The response must be terminated via a depression of the <ENTER> key. During the input of the response, you can backspace over a mistyped character via the <LEFT ARROW> key. You can clear the entire response via <SHIFT> <LEFT ARROW>. You can cancel the request via the <BREAK> key. Finally, if you wish to IMPORT the response from the display screen present prior to the invocation of CARD, you only need to depress the <CLEAR> <LEFT ARROW> keys. Export is fully described in the chapter on PRO-NTO operation.

When you invoke the CARD application, it will request PRO-NTO to open a window. If a window cannot be opened, a short beep will sound from your computer's internal speaker and CARD will terminate. This happens when you have exceeded the maximum number of windows that can be open at one time. That's a rare instance. When the window is opened, CARD will search all disk drives for a file named CARD/DAT. If you want to maintain more than one discrete card file, just keep each one on a separate floppy disk. When a CARD/DAT file is found, it will be accessed and the first card record, if any, will be displayed. If no file can be found identified as CARD/DAT, you will be prompted to specify the drive which should be used to create a CARD/DAT file via the prompt:

Create CARD/DAT on what drive?

Respond with one of your available drives (0-7).

Once the CARD/DAT file is created, the displayed window will show the record fields of the first record in the file, if any, and the CARD commands which are available. When CARD is waiting for a command entry, the cursor which is at the end of the list of available commands will be blinking [this cursor is shown as an underline "_" in the illustration which follows]. A filled in record appears as follows:



The commands permit a range of operations on the data records. A command is invoked by depressing the letter key noted by the first letter of the command word (the letter which appears capitalized). This may be entered in either upper or lower case. You can also invoke EXPORT of data when CARD is waiting for a command. Data export permits you to pass a specified field or block of data from the displayed window to the environment interrupted when you invoked CARD. More information on EXPORT can be found in the chapter on PRO-NTO operation. Details for each command follow:

ADD a record

When you wish to add a record to your card file, specify this action by depressing the <A> key. You will be presented with a window display where the contents of each field are blank. CARD will then automatically enter the EDIT command. The text field is the only field which can be edited with the entry of data. The ID field is managed via the <I>d command. The DATE and TIME fields are automatically handled by CARD. If you decide that you do not want to add the record, just depress <BREAK>. CARD will abort the addition and display whatever record was on the window at the time that ADD was invoked.

Rather than duplicate the procedures for entering your text, you are referred to the section entitled, "EDIT the currently displayed record".

Print a 3 x 5 Index CARD

This command is used to print the contents of a record or the entire data file in a format suitable for 3" high by 5" wide Index Cards in continuous forms. When you invoke CARD, you will be prompted to select the output

option via the prompt:

<A>ll, or >C<urrent?

If you want the entire data file printed, enter an <A>. If you want just the current record printed on the "card", enter the letter <C>. If you respond with no entry (by depressing just <ENTER>), the default of CURRENT will be selected. This default is indicated by the reversed angle brackets surrounding the letter "C" in the prompt. If you specify ALL records, any that have been marked with a "deleted" flag will not be printed.

You will then be asked to select the printing format via the query:

Format? >

If you just depress the <ENTER> key, you will select the default format supplied with CARD. This format is:

```
|
|
|Text line 1----------------------------
|Text line 2----------------------------
|Text line 3----------------------------
|Text line 4----------------------------
|Text line 5----------------------------
|Text line 6----------------------------
|Text line 7----------------------------
|Text line 8----------------------------
|Text line 9----------------------------
|Text line 10---------------------------
|Text line 11---------------------------
|Text line 12---------------------------
|
|[ID------]  [Date---- Time----]  [Rec #]
|
|
```

The vertical lines are not part of the output but are only illustrated here to denote the beginning of a line. Note that there are 18 lines total. This format is tailored to 18 lines per card - the quantity of lines on a 3 x 5 Index Card at six lines per inch. Each line of data will also fit on a card printed at 10 characters per inch.

The format is printed by CARD according to a string of field and printing specifiers. The string corresponding to the default format is:

';;1;[3] [4 5] [2];;;',CR

The contents of a particular field are printed when a string character is one of the field identifiers 1, 2, 3, 4, 5, or 6 [note, field 6 is currently unused]. A semicolon is interpreted as a logical carriage return. Any other

character is interpreted as a character to print. A carriage return terminates the string without taking any printing action. The text field [field]] will always be output as 40-character rows terminated by a carriage return. There will be]2 rows output. If you want to specify some other format, just enter a string of characters in response to the "Format?" prompt. A string of up to 48 characters may be entered. If you are interested in maintaining your own formats, why not keep the "strings" in a small text file? By doing that, you can interrupt the "Format?" prompt via a LIBEXEC to list your format text file then export the desired format string from the LIST display to CARD to respond to the prompt for "Format ?". You could also use a CARD record and recurse CARD to export the string back to the "Format?" prompt. Another example of nested CARD invocations shows up in the EDIT command documentation.

If you wish to abort printing at any time, just depress the <BREAK> key. When <BREAK> is detected, CARD will output an immediate carriage return and return to await a command. The window will display the record being output at the time the <BREAK> was detected.

If the printer is not available during the printing operation, CARD will display the error message,

Device not available

and terminate the CARD command. CARD will then return to await a new command.

DELETE currently displayed record

This command is used to delete the record currently being displayed. You have to confirm your request by responding to the query:

Delete: <Y> to confirm?

Any response other than the letter <Y> will abort the delete operation and return to the command prompt to await a new command. If you confirm the deletion, it is marked by placing a character value of 255D in the first character position of the ID field. This indicates a "deleted mark". A record deleted in this manner is still displayed if you come to it while "scrolling" through the card file; however, it will not be printed during a "CARD" operation when ALL records are requested nor will it be matched during a SEARCH. If you examine position 255 in the CHARSET application display, you will be able to note the "deleted" mark character displayed by your computer. Deleted records can be permanently removed from the data file by requesting the "PACK" option during a sort of the data file via the PSORT utility provided with PRO-NTO.

It is perfectly acceptable to "un-delete" a deleted record by editing the ID field and replacing the "deleted mark" with an ID character. It is not possible to restore a deleted record after a PSORT operation where you have requested the "PACK" option - at least not with any utility provided with

PRO-NTO!!!

EDIT the currently displayed record

EDIT is selected to enable you to enter additional text or modify the existing text of the current record. When EDIT is invoked, the following message is displayed in the bottom portion of the window:

Edit: ^F to file; BREAK to abort

This is an aid to letting you know that there are two ways of escaping from the edit mode. One way is to enter <CONTROL-F> to file away your edited record. The other way is the depress the <BREAK> key which will abort the editing and return to await a new command. Also, a blinking cursor will appear in the first character position of the text area. All text is entered in what is termed overstrike mode. Any ASCII non-control character will overtype the character beneath the cursor and cause the cursor to advance by one position. The following edit commands are available for your use:

Keystroke	Editing operation
<UP ARROW>	Move the cursor up one line.
<DOWN ARROW>	Move the cursor down one line.
<LEFT ARROW>	Move the cursor left one column.
<RIGHT ARROW>	Move the cursor one column to the right.
<SHIFT><LEFT ARROW>	Move to the start of the line.
<SHIFT><RIGHT ARROW>	Move to the end of the line.
<CONTROL-A>	Add a space and push rest of line right.
<CONTROL-C>	Concatenate the next line to overstrike the position of the cursor.
<CONTROL-D>	Delete the character under the cursor and move the trailing portion of the line left.
<CONTROL-F>	File the changes.
<CONTROL-S>	Split this line at the cursor and move the trailing text to a NEW next line.
<BREAK>	Cancel the changes.
<CLEAR><LEFT ARROW>	Import data.
<ENTER>	Move the cursor to the beginning of the next line.

You can insert a character by moving the cursor at the insertion position and depressing <CONTROL-A> (for "add" of a character). The characters extending from that position through to the end of the line will be shifted right and a space will be inserted. You can then overstrike the space with the desired character. To delete a character, depress <CONTROL-D> (for "delete" a character). The character at the cursor position will be deleted and all trailing characters in the line will be shifted left to "take up the slack".

The line concatenate operation invoked by <CONTROL-C> will overstrike the characters from the cursor to the end of the line with the next line. The entire next line will be deleted and all lines to the bottom of the text will be bumped up by one line. If the next line is longer than the character positions remaining, the "overflowed" characters will be dropped. If you are aware that a concatenation of the next line will overflow and you do not want this to happen, you may want to split the next line into two pieces. See the split command, <CONTROL-S>.

The split-a-line operation invoked by <CONTROL-S> will divide up the current line into two lines. This is done by first moving all lines following the current line down by one line which opens up a blank line. Next, all of the characters from the cursor through to the end of the current line are shifted to the new blank line. The old last line of the text will be dropped.

When you have completed your edits, depress <CONTROL-F> to file them away. The DATE and TIME fields will automatically be updated to reflect "today's" system date and the current system "time". These two values are initialized by you when you power up your computer or otherwise change them via the system's DATE and TIME library commands. Don't forget that if you abort the editing by depressing the <BREAK> key, the record will be left as it was prior to invoking EDIT.

As a side note, the functions of cut and paste between records or within the same record may be easily achieved by a small sequence of steps. Open up the needed space via the split command - don't worry if the text that was to be cut and pasted has been dropped. Next, invoke the CARD application recursively from the edit mode you are in. This will bring up a new invocation of card. Go to the record where the text is that you want to cut and paste. Invoke EXPORT and mark the block. This is the "cut" operation. It will be exported back to the prior CARD session into the space you have opened. Of course, you must have opened up enough space to "paste" the exported text.

Position to FIRST record

This command is used to rapidly position to the first data record of the CARD/DAT data file and display it.

GOTO a designated card

This command is provided to enable you rapid access to a card by its card number. Depress the <G> key and you will be prompted for the record number via the message,

Goto record?

Enter the decimal number of the card record you wish to display. The card numbers are displayed in the right hand portion of the card status line.

Alter the ID field

The ID field can be changed by depressing the <I> key. You will see the prompt to enter the new ID field which is:

```
ID ?
```

You can enter up to eight characters. The backspace key [<LEFT ARROW>] can be used during your entry. Any lowercase alphabetic characters will be capitalized when your entry is stored. This is to minimize field matching problems with external processing. When you terminate the ID with <ENTER>, it will update the ID field displayed in the window and automatically update the record's disk ID field as well. Note that <BREAK> will abort any changes.

Position to LAST record

This command is used to rapidly position to the last data record of the CARD/DAT data file and display it.

Position to NEXT record

This command is used to display the next data record in your card file. You can also advance to the NEXT record by depressing the <DOWN ARROW> key.

Position to PREVIOUS record

This command is used to display the previous data record in your card file. You can also position to the previous record by depressing the <UP ARROW> key.

SEARCH for a particular card

The SEARCH command is used to scan the data file starting at the record which follows the current record. The search string is entered in response to the query:

```
Search for? >
```

You can enter up to an 8-character string. If you enter a NULL string (i.e. <ENTER> by itself), the previous search string will be reused. The search string is not disturbed by intervening non-SEARCH commands. In this way, repetitive search commands can be invoked to display all records which match a single search string. The search string will be matched against each record until a match is found. The search is case insensitive. Both the text portion and the ID field will be compared to the search string. If your search string is a sub-string of the ID or text, the search will stop and that record will be displayed. The search will also stop at the last record.

If the string which you are searching is not located in any record, the message,

String not found

will be displayed. At this point, you need to depress the <ENTER> key to resume the operation of CARD.

Sorting your CARD/DAT file

You can sort your data file into ascending order according to the key string composed of the ID, DATE, and TIME fields by invoking the PSORT utility from DOS Ready. This is done via the command:

PSORT CARD/DAT [pack]

The optional parameter, "pack", is entered when you wish to remove the deleted records from your card file. The square brackets are not entered.

Disk errors

If, by chance, CARD detects an error in reading or writing your card file's records, it will display the DOS error message in the message line at the bottom of the window and then await an entry from the keyboard. This gives you an opportunity to read the error message. After you depress any key, CARD will terminate its operation and return you to the program that was running when you invoked CARD.

CARD/DAT Technical Specifications

Each record is stored in 512 bytes starting with the third sector of the CARD/DAT data file. The number of the record will be the logical record number of the record based on its LRL and is not a piece of data stored in the record. The record is composed as follows:

Field Identifier	Field Label	Width	Record Location
0	Number	0	n/a
1	TEXT	480	020-1FF
2	ID	8	000-007
3	Date [YY/MM/DD]	8	008-00F
4	Time [HH:MM:SS]	8	010-017
5	reserved	8	018-01F

The first sector of the file contains information according to the PRO-NTO general data file format. Relative bytes 0-2 contain the number of data records contained in the file (0=HSB, 1=MSB, 2=LSB). Other information used by PRO-NTO is as follows with the numbers in square brackets indicating the quantities fixed within the CARD/DAT data file (relative byte contents are stored in standard low-order high-order format):

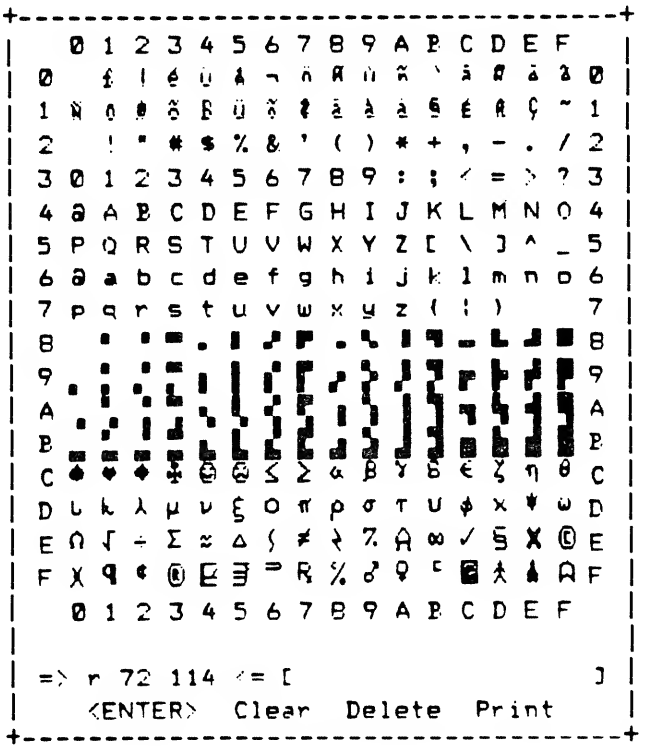
Relative	Contents
10H-11H	Logical Record Length [512D]
12H-13H	1st record of LRL treated as data [01D]
14H-15H	Relative location within each record used as the SORT key [0D]
16H	Length of the SORT key [24D]

Note: The DATE field is stored in year-month-day sequence as it is part of the sort key. In this way, ascending ordering is performed properly. The date is displayed in month-day-year sequence in the window.

Computer Character Set Application

The CHARSET application saves you from having to resort to the computer documentation manual to determine the value of a particular displayed character. CHARSET displays the entire character set - all 256 different characters - right on the screen in its own window. You can build a string of characters while within CHARSET and have it output to the printer device. Of course CHARSET doesn't have bit-mapped printer support to enable your printer to display the special characters and graphics; however, you can take the string which you have built up and EXPORT it back to a program. This gives you the perfect convenience of writing, for instance, a BASIC program and getting the special characters into a PRINT string from CHARSET, rather than from digging in your manual. CHARSET works great also to just be able to convert one-byte hexadecimal numbers to their decimal equivalent, and vice versa.

When you invoke the CHARSET application, it will request PRO-NTO to open a window. If a window cannot be opened, a short beep will sound from your computer's internal speaker and CHARSET will terminate. This happens when you have exceeded the maximum number of windows that can be open at one time. That's a rare instance. Usually when you invoke CHARSET, the window is opened and you will be presented with a window display that looks something like this:



CHARSET will have two blinking underline cursors. One will be positioned at the character position of value 0. The other will be positioned at the start of the "string" area which is at the bottom right of the window en-

closed by square brackets "[]". The character cursor may be positioned throughout the screen via the arrow keys. <LEFT ARROW> will move it left one position with the first column wrapping around to the last column. The <RIGHT ARROW> key moves the character cursor to the right by one position with the last column wrapping to the first column. The <UP ARROW> will move the character cursor up one row with the top row wrapping to the bottom row. The <DOWN ARROW> will move the cursor down one row with the bottom row wrapping to the top row. You can move to the first or last column via the <SHIFT> <LEFT ARROW> and the <SHIFT> <RIGHT ARROW> keys. <SHIFT> <UP ARROW> will position you to character value zero while <SHIFT> <DOWN ARROW> will position you to character value 255.

Wherever the cursor is positioned is considered to be the "current" character. The bottom portion of the screen being pointed to by the arrowhead will duplicate the current character blinking with the cursor. The character will be displayed along with its character value in hexadecimal followed by its value in decimal. The current character can be added to the character "string" simply by depressing the <ENTER> key. The string cursor will then be advanced by one position. The string may contain up to eighteen characters. You can print the string (output to the printer device) by depressing the letter, <P>. The string can be cleared by depressing the letter, <C>. You can delete the last string character entered by depressing the letter, <D>. Repeated deletes will continue to delete string characters until no more remain (the same as <C>lear). Don't forget that you can EXPORT the string (or anything else on the window) back to the program environment which was interrupted when you invoked CHARSET. In this way, you can "build" up a string for use in a BASIC program PRINT statement or a C-program's PUTS() display function.

CHARSET allows you to toggle the display to the alternate character set by depressing the <T> key. This function actually switches your machine to the alternate set by sending a character value of 22 (decimal) to the display device. Note that all special characters currently displayed on the screen will be toggled to their alternate character display - even any in the string field.

DIALER Application

The DIALER application will maintain a small data file of records containing information useful for keeping a telephone list. The data items available within each DIALER record are identified in the following table:

Field Identifier	Field Label	Width	Permissable Entries
0	ID	8	20H-7FH
1	Telephone #	16	20H-7FH
2	Description	40	20H-7FH

The ID field is used to specify a key string for searching and sorting the records. The Telephone Number is the string which specifies what is sent to the modem for autodialing purposes. This would, of course, also be the telephone number of the individual or entity identified by the record. The Description field gives you up to forty character positions for adding any note you need to maintain with the telephone record.

DIALER has a few commands that display a prompt message and expect a response. The response must be terminated via a depression of the <ENTER> key. During the input of the response, you can backspace over a mistyped character via the <LEFT ARROW> key. You can clear the entire response via <SHIFT> <LEFT ARROW>. You can cancel the request via the <BREAK> key. Finally, if you wish to IMPORT the response from the display screen present prior to the invocation of DIALER, you only need to depress the <CLEAR> <LEFT ARROW> keys. Export is fully described in the chapter on PRO-NTO operation.

When you invoke the DIALER application, it will request PRO-NTO to open a window. If a window cannot be opened, a short beep will sound from your computer's internal speaker and DIALER will terminate. This happens when you have exceeded the maximum number of windows that can be open at one time. That's a rare instance. When the window is opened, DIALER will next make sure that the COM/DVR serial driver is installed as this is required for autodialing purposes. If COM is not available, DIALER will beep twice then abort.

DIALER will search all disk drives for a file named DIALER/DAT which is used to contain the telephone list records. If you want to maintain more than one discrete telephone list file, just keep each one on a separate floppy disk. When a DIALER/DAT file is found, it will be accessed and the first four dialer records, if any, will be displayed. If no file can be found identified as DIALER/DAT, you will be prompted to specify the drive which should be used to create a DIALER/DAT file via the prompt:

Create DIALER/DAT on what drive?

Respond with one of your available drives (0-7). Once the DIALER/DAT file is created, the displayed window will show the record fields and DIALER commands which are available.

The autodialer application will support the automatic dialing of a number from any Hayes compatible modem. It also provides a complement of commands which are used to maintain the telephone list(s). It will display a window such as the following sample window when DIALER opens the DIALER/DAT file and is ready for use:

```
+-------------------------------------------------------------------------+
|     [  ID  ][ Phone Number ][              Description               ]  |
| ==> MISOSYS   450-4181      MISOSYS Work Phone (Roy Soltoff)            |
|     COMPU     352-7500      Compuserve                                  |
|     TCUG      836-0384      TCUG Resource Network                       |
|     PWP       441-8212      Perfect Word Processing                     |
|-------------------------------------------------------------------------|
| _ Add   Call   Delete   Edit   First   Input   Last   Macro   Search    |
+-------------------------------------------------------------------------+
```

The current record is identified by the arrow pointer "==>". Use the <UP ARROW> and <DOWN ARROW> keys to reposition this pointer to different records. When DIALER is waiting for a command entry, a cursor is blinking in the bottom portion of the window which displays the list of available commands. This cursor is noted in the illustration by the underline "_" to the left of "Add". The following commands are supported:

Command	Purpose of command
Add	Add a new entry
Call	Call the current entry
Delete	Delete the current entry
Edit	Edit the current entry
First	Position to first entry
Input	Input a number manually to call
Last	Position to last entry
Macro	Edit macro storage strings
Search	Search for a particular ID

These commands permit a range of operations on the data records. A command is invoked by depressing the letter key noted by the first letter of the command (the letter which appears capitalized in the menu). This may be entered in either upper or lower case. You can also invoke EXPORT of data when ADDRESS is waiting for a command. Data export permits you to pass a specified field or block of data from the displayed window to the environment interrupted when you invoked ADDRESS. More information on EXPORT can be found in the chapter on PRO-NTO operation.

Concept of Autodialing

Before we go into the details on each command, it is best to provide an overview of the concepts associated with autodialing and the type of support offered by DIALER. To begin with, you may have noticed that DIALER provides a maximum field length of 16 characters for the telephone number. If you are

using a long distance service or a PBX service which requires the entry of extra digits, you may not appreciate having only 16 positions for the storage of the telephone number. Well, we did not want to leave you out in the cold. DIALER supports 15 macro keys designated A, B, C, ... M, N, O. Each macro key can have up to 16 characters associated with it. Thus, constant sequences of digits and codes can be stored as macros. For instance, you may be using a service which requires a local access number [i.e. 555-1234] to be followed up with a six-digit access code [i.e. 123456] after the second dial tone is detected. This can be coded, for instance, as macro-A using the string, "5551234,T123456". The comma happens to be the code used by the Hayes modem to wait for a second dial tone. The "T" indicates that tone dialing is to commence from that point on. Then prefix your record number with "A".

If, by chance, you happen to be using a Hayes-compatible modem which uses a function letter or letters the same as a macro letter, just reserve a macro to have as its string, the letter needed by the modem. Say, for instance, that the letter "D" is needed. You could just edit macro-D to be the letter "D". Any letter that is part of a macro is transmitted to the modem. This means that macros do not nest.

One other operation that is significant is the initialization of your modem. When DIALER sends a character to the modem to initiate a call, it prefixes it with the three characters, "ATD". DIALER provides one additional macro that will always be sent to your modem whenever DIALER is invoked. Note that it is NOT sent to the modem each time a call is placed but ONLY when DIALER is invoked. This macro is designated as macro-@ for the purposes of editing; however, it never is entered into a telephone number string. Macro-@ could be used to set up the "P" required by pulse only telephone lines. It could also be used to set the wait-for-dial-tone time [i.e. S6=5]. For additional details concerning the dialing strings needed by your modem, consult your modem's owner's manual.

ADD a record

When you wish to add a record to your telephone file, specify this action by depressing the <A> key. You will be presented with a display where the arrowhead pointer is pointing to a blank record. The record is free form and the three field entries are treated as one long string. Just enter the proper information into the field positions marked by the descriptions within square brackets. You can use the same editing keys described for the Edit command. If you decide that you do not want to add the record, just depress <BREAK>. DIALER will abort the addition. When you are finished typing in the record, depress <CONTROL-F> to file away your addition. Note that depression of the <ENTER> key will also file the record.

CALL the current entry

This command is used to autodial the record currently being pointed to by the arrow pointer. The message,

Dial complete, H to hangup

will be displayed as the number is dialed. As soon as you hear the ringing from the modem's speaker, you can lift the telephone receiver off hook and depress the "H" key to "hangup" the modem.

DELETE currently displayed record

This command is used to delete the record currently being displayed. You have to confirm your request by responding to the query:

Delete: <Y> to confirm?

Any response other than the letter <Y> will abort the delete operation and return to the command prompt to await a new command. If you confirm the deletion, it is marked in the following manner. A character value of 255D is placed in the first character position of the ID field and the record is filed. This indicates a "deleted mark". A deleted record is still displayed if you come to it while "scrolling" through the telephone list; however, a deleted record will not be matched during a SEARCH. If you examine position 255 in the CHARSET application display, you will be able to note the "deleted" mark character displayed by your computer. Deleted records can be permanently removed from the data file by requesting the "PACK" option during a sort of the data file via the PSORT utility provided with PRO-NTO.

It is perfectly acceptable to "un-delete" a deleted record by editing the 1st position and deleting the "deleted mark". It is not possible to restore a deleted record after a PSORT operation where you have requested the "PACK" option - at least not with any utility provided with PRO-NTO!!!

EDIT the currently displayed record

EDIT is selected to enable you to change any of the data in any field. When EDIT is invoked, the following message is displayed in the bottom portion of the window:

Edit: ^F to file; BREAK to abort

This is an aid to letting you know that there are two ways of escaping from the edit mode. One way is to enter <CONTROL-F> to file away your edited record. The other way is the depress the <BREAK> key which will abort the editing and return to await a new command. Also, a blinking cursor will appear in the first character position of field zero (ID). All text is entered in what is termed overstrike mode. Any ASCII non-control character will overtype the character beneath the cursor and cause the cursor to advance by one position. The following edit commands are available for your use:

Keystroke	Editing operation
<LEFT ARROW>	Move the cursor left one column.
<RIGHT ARROW>	Move the cursor one column to the right.
<SHIFT><LEFT ARROW>	Move to the start of the line.
<SHIFT><RIGHT ARROW>	Move to the end of the line.
<CONTROL-A>	Add a space and push rest of line right.
<CONTROL-D>	Delete the character under the cursor and move the trailing portion of the line left.
<CONTROL-F>	File the changes.
<BREAK>	Cancel the changes.
<CLEAR><LEFT ARROW>	Import data.
<ENTER>	Same as <CONTROL-F>

You can insert a character by moving the cursor at the insertion position and depressing <CONTROL-A> (for "add" of a character). The characters extending from that position through to the end of the line will be shifted right and a space will be inserted. You can then overstrike the space with the desired character. To delete a character, depress <CONTROL-D> (for "delete" a character). The character at the cursor position will be deleted and all trailing characters in the line will be shifted left to "take up the slack". The <LEFT ARROW> and <RIGHT ARROW> keys will reposition the cursor across the record.

When you have completed your edits, depress <CONTROL-F> to file them away. Don't forget that if you abort the editing by depressing the <BREAK> key, the record will be left as it was prior to invoking EDIT.

INPUT a number manually to call

This command will allow you to use the modem's dialing facility to dial a number which you manually type in (or import from a previous screen). DIALER will prompt you for the number via the message:

Number to dial ?

You can enter up to a maximum of 16 characters. Macro keys are acceptable as part of the input.

Position to FIRST record

This command is used to rapidly position to the first data record of the DIALER/DAT data file and display it.

Position to LAST record

This command is used to rapidly position to the last data record of the DIALER/DAT data file and display it.

MACRO key editing

When you want to change the contents of a macro key or keys, use this command. It will display the prompt,

Edit macro < @ - O > ?

If you want to edit the string which is always output to the modem each time DIALER is invoked, enter the "@". The string will be presented for your editing convenience. You can select any of the other 15 macros for editing by depressing the appropriate letter key, "A" through "O". The string is edited via overstrike; however, you can move the cursor to each position via the <LEFT> or <RIGHT> <ARROW> keys. File changes via <ENTER> or <Control-F>.

Position to next record

This command is used to display the next data record in your telephone list file. You advance to the NEXT record by depressing the <DOWN ARROW> key.

Position to previous record

This command is used to display the previous data record in your telephone list file. You position to the previous record by depressing the <UP ARROW> key.

SEARCH for a particular record

The SEARCH command is used to scan the data file starting at the record which follows the current record. The search string is entered in response to the query:

Search for? >

You can enter up to an 8-character string. If you enter a NULL string (i.e. <ENTER> by itself), the previous search string will be reused. The search string is not disturbed by intervening non-SEARCH commands. In this way, repetitive search commands can be invoked to display all records which match a single search string. The search string will be matched against each record until a match is found. The ID field of the record is a single 8-character key field. If your n-character search string matches the first n characters of the ID field, the search will stop and that record will be displayed. The search will also stop at the last record.

Sorting your DIALER/DAT file

You can sort your data file into ascending order according to the key string ID field by invoking the PSORT utility from DOS Ready. This is done

via the command:

```
PSORT DIALER/DAT [Pack]
```

The optional parameter, "Pack", is entered when you wish to remove the deleted records from your dialer file. The square brackets are not entered.

Disk errors

If, by chance, DIALER detects an error in reading or writing your dialer file's records, it will display the DOS error message in the message line at the bottom of the window and then await an entry from the keyboard. This gives you an opportunity to read the error message. After you depress any key, DIALER will terminate its operation and return you to the program that was running when you invoked DIALER.

DIALER/DAT Technical Specifications

Each record is stored in 64 bytes starting with the third sector of the DIALER/DAT data file. The record is composed as follows:

Field Identifier	Field Label	Width	Record Location
0	ID	8	000-007
1	Telephone #	16	008-017
2	Description	40	018-03F

The second file sector stores the 16-byte macro strings. The first file sector contains information according to the PRO-NTO general data file format. Relative bytes 0-2 contain the number of data records contained in the file (0=HSB, 1=MSB, 2=LSB). Other information used by PRO-NTO is as follows with the numbers in square brackets indicating the quantities fixed within the DIALER/DAT data file (relative byte contents are stored in standard low-order high-order format):

Relative	Contents
10H-11H	Logical Record Length [64D]
12H-13H	1st record of LRL treated as data [08D]
14H-15H	Relative location within each record used as the SORT key [0D]
16H	Length of the SORT key [08D]

Note: DIALER can be invoked from the COMM program provided with your DOS; however, if you do this while you are receiving a character string from the communications line, any characters received by the modem while you are in DIALER will be lost. Thus, DIALER should be used from COMM only when you get into COMM and want to dial a communications link number.

DOSAVE Application

The DOSAVE application will allow you to save the 80 column by 24 row video screen display into a disk file as 24 80-byte strings, each terminated by a carriage return. The complete assembly language source code used to generate DOSAVE/APP is included as a programming illustration for those folks wishing to learn more about writing applications in assembler. The source file specification is DOSAVE/ASM and is in a format usable by the PRO-CREATE macro assembler available from MISOSYS.

When you invoke the DOSAVE application, it will request PRO-NTO to open a window. If a window cannot be opened, a short beep will sound from your computer's internal speaker and DOSAVE will terminate. This happens when you have exceeded the maximum number of windows that can be open at one time. That's a rare instance. When the window is opened, DOSAVE will prompt you to enter a file specification via the prompt,

Filespec?

The screen image that existed prior to the display of the DOSAVE window will be copied to the disk file which was specified in response to the prompt.

Although the DOSAVE applicatiion may appear simplistic, it is nevertheless useful in certain situations. If you think about it, it could be useful to capture a large screenful of data from one application for input to another. The disk file would then be used as a temporary holding file.

A Mini-Terminal

The terminal application is a very simple terminal program. It requires the use of the COM/DVR serial driver installed as the "*CL" device. The only input filtering performed is the conversion of a form feed [X'0C'] into a clear screen function. The only output filtering is the conversion of the <BREAK> key to a modem break.

When you invoke the TERM application, it will request PRO-NTO to open a window. If a window cannot be opened, a short beep will sound from your computer's internal speaker and TERM will terminate. This happens when you have exceeded the maximum number of windows that can be open at one time. That's a rare instance. When the window is opened, TERM will next make sure that the COM/DVR serial driver is installed as the "*CL" device as this is required for communications purposes. If COM is not available, TERM will beep twice then abort.

TERM is the only application supplied with PRO-NTO that does not use the <BREAK> key to exit the application. That's to permit <BREAK> to emit a modem break to the communications line. TERM will only respond to a command of <CLEAR> <SHIFT> <=> to exit. This is the same keystroke combination as used in the COMM communications program.

Since TERM is a PRO-NTO application, it can be used from practically any program; thus, you have terminal communications capability available at the touch of a button.

Typewriter

The TYPER application has been provided as a small example of what can be easily done with PRO-NTO. Its assembly language source code is documented in the chapter, "For the programmer". TYPER, however, can be a useful little tool to give you the capability of typing directly to your printer - with visual line buffering on the video display so you can see what you are typing.

When you invoke the TYPER application, it will request PRO-NTO to open a window. If a window cannot be opened, a short beep will sound from your computer's internal speaker and TYPER will terminate. This happens when you have exceeded the maximum number of windows that can be open at one time. That's a rare instance. When the window is opened, TYPER provides you with a two-line 80-character window where you can type characters a line at a time. When you depress the <ENTER> key, the line which you have typed will be sent to your on-line printer. You then can continue to enter lines which will be sent to your printer until you depress the <BREAK> key.

While you are inputing a line, you can BACKSPACE over a mistake by using the <LEFT ARROW> key. You can "power-delete" the entire line via <SHIFT> <LEFT ARROW>. You can also use IMPORT [<CLEAR> <LEFT ARROW>] to obtain a line or lines of "keystrokes" from the video screen which was displayed prior to entering TYPER.

General Information

This chapter is devoted to providing the programmer with specifications and standards on using the PRO-NTO SuperVisor Call both from within general programs and PRO-NTO applications. The term program differs from the term application in that applications are programs designed and written in a certain manner which as to permit their operation solely within the memory space ranging from 2400H through 2FFFH inclusive and further providing that the program code portion of the program module is restricted to the code space 2800H through 2FFFH inclusive. Furthermore, applications are resident on disk as a core-image executable load module. Any program can make use of the window control function of PRO-NTO; however, only applications can be invoked by the application manager of PRO-NTO or by the PRUN facility.

The first topic of discussion will be the use of the window SuperVisor Call, SVC-124. When PRO-NTO is installed, it attaches the window controller to the DOS via the SVC. The SVC has many functions associated with it in a manner similar to the @VDCTL SVC (SVC-15) provided by the DOS. These functions generally correspond to their @VDCTL counterpart. Additional functions have been added which correspond to @KEYIN, @DSP, and @DSPLY SVCs.

Window Supervisor Call Facility

The following sections show the use and register setup associated with each function of the Window Supervisor Call, @WINDOW - SVC 124

WINDOW KEYIN - Function 0

This SVC function is used to perform a line input while blinking the cursor. The function will pass back status concerning the depression of the <BREAK> key, the EXPORT request, and the number of characters input. If register C is set to zero, then an immediate return after one character will be made. This is somewhat like the DOS @KEY SVC except that WINDOW KEYIN will blink the cursor and display the character input.

```
Registers Affected: AF, BC, DE.
 B => 0; Request a line or character of input.
 C => Number of characters to accept.
      (See note if register C is equal to 0)
HL => Start of your keyin buffer
 B <= Actual number of characters input if Z return
HL <= Start of buffer
 A <= First character in input string, or error code if NZ return
CF <= Carry flag set if input terminated by <BREAK> or EXPORT
 C <= If Carry flag set, will contain 00 if <BREAK> depressed,
      or 01 if EXPORT depressed. Register C is indeterminate
      if Carry flag not set or if NZ return.
 Z <= Set if no error, else reset.
```

Note 1: If function is requested with C equal to 0, then WKEYIN will return after one keystroke; all character values are accepted. Also, register pair HL will NOT be used to store the character input.

Note 2: Unlike DOS @KEYIN routine, this routine does not echo CR's, does not accept tabs, or any character less than x'20' or greater than x'7F'. This routine WILL, however, blink the cursor.

WINDOW "PEEK"

This SVC function will peek the character at the current window cursor location.

Registers Affected: AF, BC, DE.
B => 1; Gets the character at the position identified by HL.
HL => Contains the window row (0-23) in register H,
and window column (0-79) in register L.
A <= Will be returned with the character at "HL".
Z <= Set if the operation was successful.

WINDOW "POKE"

This SVC function will poke a character into the window at the current window cursor location.

Registers Affected: AF, BC, DE.
B => 2; Puts the character at the position identified by HL.
HL => Contains the window row (0-23) in register H,
and window column (0-79) in register L.
C => Contains the character to put at "HL".
Z <= Set if the operation was successful.

SET THE WINDOW CURSOR POSITION

This SVC function will set the window cursor to a program selected position.

Registers Affected: AF, B, DE.
B => 3; Moves the window cursor to the positon
identified by HL.
HL => Contains the window row (0-23) in register H,
and window column (0-79) in register L.
A <= Will contain the error code if an error was encountered.
Z <= Set if the operation was successful.

OBTAIN THE WINDOW CURSOR POSITION

This SVC function will obtain the position of the cursor within the window.

```
Registers Affected: AF, B, DE.
 B => 4; Obtains the current window cursor position
      by row and column.
HL <= Contains the window row (0-23) in register H,
      and window column (0-79) in register L.
 A <= Will contain the error code if an error was encountered.
```

BUFFER TO WINDOW

This SVC function will transfer a video image contained in a block of RAM to the window.

```
Registers Affected: AF, BC, DE, HL.
 B => 5; Moves a block of RAM to the window.
HL => A pointer to the user's RAM block.
 A <= Will contain the error code if an error was encountered.
 Z <= Set if the operation was successful.
```

Note 1: This is a warning! The buffer may NOT reside from 2800H to 2850H.

WINDOW TO BUFFER

This SVC function will copy the window image to a block of RAM.

```
Registers Affected: AF, BC, DE, HL.
 B => 6; Moves the window to a block of RAM.
HL => A pointer to the user's RAM block.
 A <= Will contain the error code if an error was encountered.
 Z <= Set if the operation was successful.
```

Note 1: This is a warning! The buffer may NOT reside from 2800H to 2850H.

CREATE A WINDOW

This SVC function will create a new window. If the maximum depth level has been reached prior to the request, the SVC will return with the Z-flag reset. You must test this flag to ensure that the window was in fact opened.

```
Registers Affected: AF, BC, DE, HL.
 B => 7; Creates a window based on the specifications
      in register pairs HL and DE.
HL => Contains the screen row (0-23) in register H,
      and screen column (0-79) in register L for the
```

upper left corner of the window.
DE => Contains the number of window rows (1-24) in register H,
and the number of window columns (1-80) in register L.
A <= Will contain the error code if an error was encountered.
Z <= Set if the operation was successful.

CLOSE A WINDOW

This SVC function is used to close the last window opened via CREATE. Any application which opens a window must close it before returning. You have been warned! If register-C is set non-zero, then WCLOSE will perform the function of export. Note that the operation of EXPORT will provide its own RET during the close operation and the SVC will not return to the caller.

Registers Affected: AF, BC, DE, HL.
B => 8; Closes the window.
C => export flag; <>0 then export requested,
0 then no export.

DISPLAY A CHARACTER IN THE WINDOW

This SVC function will display a character at the current window cursor location and advance the cursor. It is similar to the @DSP SVC, not WPOKE!

Registers Affected: AF, BC.
B => 9; Display a character in the window.
C => Contains the character to display.

DISPLAY A STRING IN THE WINDOW

This SVC function will display a string at the current window cursor position. It is similar to the DOS @DSPLY SVC. The string must be terminated by either ETX or CR.

Registers Affected: AF, BC.
B => 10; Display the string pointed to by HL in the window.
HL => Points to the 1st byte of the string.
HL <= Points to the 1st byte of the string.

WINDOW IMPORT

This SVC function is used to perform the function of IMPORT. When invoked, the previous screen will be restored temporarily and the IMPORT cursor will be presented. After the block of text is marked, PRO-NTO will restore the application's window and return from the SVC. Thereafter, calls to @KEY or WINDOW KEYIN, will be satisfied from the marked text until the text is exhausted, at which time key requests will revert to the physical

keyboard.

Registers Affected: AF, BC, DE, HL.
B => 11; Request import from previous screen.

Window Display Driver Control Code Support

The following control character codes are supported by the window display driver:

Code	Value	Functional Operation
ETX	x'03'	End of text. terminates string, no linefeed.
BKSP	x'08'	Backspace and erase.
LF	x'0A'	Linefeed.
CR	x'0D'	Carriage return and linefeed.
LEFT	x'18'	Move cursor left.
RIGHT	x'19'	Move cursor right.
DOWN	x'1A'	Move cursor down.
UP	x'1B'	Move cursor up.
HOME	x'1C'	Home cursor.
START	x'1D'	Home to column zero.
EREOL	x'1E'	Erase to end of line.
EREOW	x'1F'	Erase to end of window.

Any control code not supported by the window driver will be passed through to the DOS video driver. Since the previous screen image outside of the window border remains on the screen, you may want to avoid using inverse video. PRO-NTO does not save the state of the inverse video setting when creating or closing a window. That's another reason you may want to avoid using inverse video.

The PRO-NTO window display driver fully supports window scrolling. There is no support for window scroll protect.

The following error codes are supported by the window display driver:

Device not available - error 8
SVC parameter error - error 43

SVC Macros - WINDOW/MAC

A macro file named WINDOW/MAC has been provided which supports the SVC invocation when used with the PRO-CREATE macro assembler available from MISOSYS. In addition to the window control macros, it defines the two key code values for export and import as follows:

```
EX_PORT EQU     89H             ;<CLEAR><RIGHT>
IM_PORT EQU     88H             ;<CLEAR><LEFT>
```

The available macro calls and their parameters, if any, are as follows:

@WKEYIN #LEN,#BUF

@WKEYIN provides two optional parameters, #LEN and #BUF. If only one parameter is provided, it is assumed to be #LEN.

#LEN - the maximum number of characters allowed for input
#BUF - a 16-bit pointer to the keyin buffer

```
@WKEYIN MACRO   #LEN,#BUF
        IFEQ    %%,0
        @WINDOW 0
        ELSE
        IFEQ    %%,2
        LD      HL,#BUF
        ENDIF
        LD      BC,#LEN
        @WINDOW
        ENDIF
        ENDM
```

@WPEEK #ROW,#COL

@WPEEK provides two optional parameters. If one is entered, the other must be also.

#ROW - what row to peek
#COL - what column to peek

```
@WPEEK  MACRO   #ROW,#COL
        IFEQ    %%,2
        LD      HL,#ROW.SHL.8.OR.#COL
        ENDIF
        @WINDOW 1
        ENDM
```

@WPOKE #ROW,#COL

@WPOKE provides two optional parameters. If one is entered, the other must be also.

#ROW - what row to poke
#COL - what column to poke

```
@WPOKE  MACRO    #ROW,#COL
        IFEQ     %%,2
        LD       HL,#ROW.SHL.8.OR.#COL
        ENDIF
        @WINDOW  2
        ENDM
```

@WSCUR #ROW,#COL

@WSCUR provides two optional parameters. If one is entered, the other must be also.

#ROW - what row to set
#COL - what column to set

```
@WSCUR  MACRO    #ROW,#COL
        IFEQ     %%,2
        LD       HL,#ROW.SHL.8.OR.#COL
        ENDIF
        @WINDOW  3
        ENDM
```

@WGCUR

@WGCUR has no parameters.

```
@WGCUR  MACRO
        @WINDOW  4
        ENDM
```

@WB2W #BUF

@WB2W provides one optional parameter.

#BUF - 16-bit pointer to image buffer

```
@WB2W   MACRO    #BUF
        IFEQ     %%,1
        LD       HL,#BUF
        ENDIF
```

```
        @WINDOW  5
        ENDM
```

@WW2B #BUF

@WW2B provides one optional parameter.

#BUF - 16-bit pointer to image buffer

```
@WW2B   MACRO    #BUF
        IFEQ     %%,1
        LD       HL,#BUF
        ENDIF
        @WINDOW  6
        ENDM
```

@WCREAT #SROW,#SCOL,#NROW,#NCOL

@WCREAT provides four optional parameters. If one is entered, they all must be entered.

#SROW - northwest row of window
#SCOL - northwest column of window
#NROWS - number of rows of window
#NCOLS - number of columns of window

```
@WCREAT MACRO    #SROW,#SCOL,#NROW,#NCOL
        IFEQ     %%,4
        LD       HL,#SROW.SHL.8.OR.#SCOL
        LD       DE,#NROW.SHL.8.OR.#NCOL
        ENDIF
        @WINDOW  7
        ENDM
```

@WCLOSE

@WCLOSE has no parameters.

```
@WCLOSE MACRO
        @WINDOW  8
        ENDM
```

@WDSP

@WDSP has no parameters.

```
@WDSP   MACRO
        @WINDOW  9
        ENDM
```

@WDSPLY #STRING

@WDSPLY provides one optional parameter.

#STRING - 16-bit pointer to the message string

```
@WDSPLY MACRO    #STRING
        IFEQ     %%,1
        LD       HL,#STRING
        ENDIF
        @WINDOW  10
        ENDM
```

@WIPORT

@WIPORT has no parameters.

```
@WIPORT MACRO
        @WINDOW  11
        ENDM
```

@WINDOW #FUNCT

This is a miscellaneous macro which can be used when either register B already contains the window function number or is passed as a parameter.

```
@WINDOW MACRO    #FUNCT
        IFEQ     %%,1
        LD       B,#FUNCT
        ENDIF
        LD       A,124
        RST      40
        ENDM
```

$DS #LABEL,#LENGTH

This macro can be used to define data fields in the second half of the data area (2600H-27FFH). Two parameters are required. The macro will automatically define a data counter to begin at address 2600H. Thereafter, the data counter will be advanced according to the length of the data field.

#LABEL - This is the symbol to be defined
#LENGTH - This is the amount of storage to be reserved for #LABEL

```
$DS      MACRO    #LABEL,#LENGTH
         IFNDEF   DATA$
DATA$    DEFL     2600H
         ENDIF
#LABEL   EQU      DATA$
DATA$    DEFL     DATA$+#LENGTH
         ENDM
```

Window Control Interface from C Programs

A C library of functions called WINDOW/CCC has been provided to enable the easy access of the window controller from programs written in C. The use of this library is documented in a section on the WINDOW/CCC library.

Window Control Interface from High-Level Language Programs

A device driver named WINLINK has been provided to enable a high-level language program to easily interface to the window controller. The use of this device driver is documented in the section on WINLINK.

Application Format and Protocol

All PRONTO applications must adhere to a strict protocol in order to be able to work with the PRO-NTO application manager. To begin with, the application must be named with the file extension "/APP". The file itself must conform to the following format.

1. The first sector will contain the following data:

   bytes 00H-05H: Must contain the string "PRONTO" to indicate that the file contains a PRONTO application.

   bytes 06H-12H: Description of application to be inserted into the PRONTO menu if the application is one of the four memory resident applications. The string is terminated by an ETX.

   bytes 13H-DFH: Reserved for future expansion.

   bytes E0H-FFH: May contain a copyright message, blank otherwise.

2. The remaining sectors must contain a core image to be loaded from address 2800H up to a maximum of 2FFFH.

The application must be coded so as to conform to the following:

1. After an application has been loaded, control will be transferred to address 2800H; thus, 2800H must be the code entry point.

2. Terminate the application by issuing a RET instruction after any open window has been closed.

3. It is an application's responsibility to close any windows which may have been created during its execution. Failure to do so will cause unpredictable results [it is actually predicted that PRO-NTO will remain activated].

4. The region from 2400H through 27FFH inclusive has been saved by the PRO-NTO application manager and is available for your use as a data region.

5. Finally, PRO-NTO provides a stack space of approximately 250 bytes. Do not change the stack pointer in your application to any address within the range 2400H-2FFFH as application recursion may wipe out your stack. The PRO-NTO stack area should provide space sufficient for all reasonable applications.

The following assembly listing is a very small application which illustrates some of the routines standardized for PRO-NTO applications.

```
;TYPER - a PRONTO application allowing direct access to the system printer
;
;       by Scott A. Loomer, 12 March 1985
;       with over-the-shoulder-assistance from Karl Hessinger
;
;       demos a very simple PRONTO application
;
;       Use: when invoked, type what you want (<80 chars) and upon
;       hitting RETURN, it is sent to the printer. Import is supported,
;       so you can mark a block of text and the entire block is printed.
;
@WKEYIN EQU     0
@WCREAT EQU     7
@WCLOSE EQU     8
@WDSP   EQU     9
@PRINT  EQU     14
@SOUND  EQU     104
@WINDOW EQU     124
;*=*=*
;       Establish macro
;*=*=*
SVC     MACRO   #NUM
        LD      A,#NUM
        RST     40
        ENDM
;
        OPTION  CI              ;PRO-NTO applications are core image
;
;*=*=*
;       The following is the header protocol
;*=*=*
        ORG     2700H           ;Forces START to 2800H
        DB      'PRONTO'
        DB      'Typewriter ',3
        DC      .HIGH.$.SHL.8-$+256,0
;*=*=*
;       This is where the program starts
;*=*=*
        IFNE    $,2800H
        ERR     'Something is wrong in the header
        ENDIF
START   LD      B,@WCREAT       ;Function to open a window
        LD      HL,16.SHL.8+0   ;Origin NW corner at row 16, col 0
        LD      DE,2.SHL.8+80   ;Window to have 2 rows, 80 columns
        SVC     @WINDOW
        JR      Z,OK            ;Go if we were able to open it
        LD      B,00.SHL.4+02   ;No good, set up parms for beep
```

```
        SVC     @SOUND          ;Beep the speaker and return
        RET
;*=*=*
;       Window opened, proceed to accept lines until BREAK
;*=*=*
OK      LD      HL,BUFFER       ;Point to key buffer
        LD      BC,@WKEYIN.SHL.8+79
        SVC     @WINDOW         ;Accept keyin of up to 79 chars
        JR      NC,OK2          ;Go if not BREAK or EXPORT
        LD      B,@WCLOSE       ;Close the window (reg C will
        SVC     @WINDOW         ;  be set for BREAK or EXPORT code
        RET
;*=*=*
;       Print the line entered
;*=*=*
OK2     SVC     @PRINT          ;HL still points to buffer
        LD      BC,9.SHL.8+13   ;Display a CR
        SVC     @WDSP
        JR      OK              ;Loop for next line
;*=*=*
;       This is the key buffer
;*=*=*
BUFFER  DS      80
;*=*=*
;       Standard memory overflow check
;*=*=*
        IF      $.GT.3000H              ;Check on memory overflow
        ERR     'Memory overflow!!!'
        ENDIF
;*=*=*
;       Zero last sector of application file - patch space
;*=*=*
        IFLT    $,3000H
        DC      .HIGH.$.SHL.8-$+256,0   ;Zero remainder of sector
        ENDIF
        END     START                   ;START must be 2800H
```

To further aid you in your study of programming applications, the source code to the DOSAVE application is included on the PRO-NTO release diskette. Study this application well. It illustrates most of the fine points of writing PRO-NTO applications. Finally, both the chapter on WINLINK and the chapter on the WINDOW/CCC library contain high level language programming examples of programs which access the window controller.

Extensive use has been made of the PRO-CREATE (EDAS) macro assembler in developing the entire PRO-NTO package. This macro assembler provides the tools to easily create the core image formats required for PRO-NTO applications.

An Aid to Debugging Applications

PRO-NTO provides a facility to invoke an application in a DEBUG environment as part of the PRUN utility. By specifying the command,

PRUN filename:d (DEBUG)

PRUN will load your application and transfer control to the system's DEBUG module. The program counter will be established at address 27FFH, one byte prior to the application's ENTRY point. At this point, use DEBUG's "J" command to step to 2800H and commence your debugging.

If you need to do a lot of debugging of your application programs, you may want to consider the use of the DD DEBUG Disassembler module which is part of the PRO-DD&T package available from MISOSYS. DD enhances the system DEBUG by including a disassembly of the current instruction in the display.

Finally, if you are having trouble in programming an application, our support staff may be able to help. Telephone calls may be impractical as your code will most likely need to be examined. Your best bet is to send us a floppy disk with the source code and any peripheral files you use as well as an assembly source printer listing. If you are not a programmer but would like us to program an application according to your specifications, please provide your complete specifications in writing to us.

What is WINDOW/CCC?

The PRO-NTO window manager is controlled via a DOS SuperVisor Call (SVC) facility which is attached to the resident DOS when PRO-NTO is installed. Access to the SVC is most normally done by machine language routines; however, the window manager may be controlled from any programming language. If you are programming in "C", you could access the window SVC via the LC call() function provided in the installation library. However, you may find it more convenient to control windows by utilizing the WINDOW/CCC C-function library provided with the PRO-NTO package.

WINDOW/CCC is a library of window interfacing functions designed to be compatible with the PRO-LC C-compiler available from MISOSYS, Inc. The library can be "#included" with your C-source program and compiled whenever you compile your program. The library can also be precompiled then included as an "/ASM" file via the #asm ... #endasm construct and the *GET facility of the assembler. It could also be compiled and the assembly file(s) placed into a /LIB library (using PRO-PaDS) to be searched during the assembly of LC/ASM. The source code for each function is included in the information for each function call. The entire library is included as the file named, WINDOW/CCC. The following #define statements are part of the library:

```
#define AF    0
#define BC    1
#define DE    2
#define HL    3
```

which permit an understandable method of coding the SVC interfacing.

Controlling the Window Manager

The remaining sections will show you how to control the window manager from a C-language program so that you can create windows and use them. The library supports thirteen functions of communications with the window manager. These functions are: keyin input, peek, poke, set cursor position, obtain cursor position, copy image buffer to window, copy window to image buffer, create a window, close a window, put a character, put a string, export a block, and import a block; all functions referencing operation on the window. The following sections explain the use of the thirteen functions.

Window Keyin - Function wgets()

The wgets() function is used to perform a line input while blinking the cursor [so you can tell where it is if you have not otherwise turned it on]. It is somewhat like C's gets() function except that WINDOW KEYIN will always blink the window cursor and display the character input. The function will provide a return code status indicating the character used to terminate the input. This concerns itself with the depression of <ENTER>, <BREAK>, or EXPORT during the keyin. The return code will be an integer value of zero,

one, or two if respectively <ENTER>, <BREAK>, or EXPORT terminated the keyed in response. If IMPORT is keyed during the key in response, PRO-NTO will automatically handle the request within the WKEYIN processing.

When you invoke wgets(), two arguments are required. First, the key buffer must be a minimum size equal to the maximum number of characters plus one. Second, you must provide a count of the maximum number of characters permitted for input as a parameter. This is desirable in the case of windows, thus, it behaves like C's fgets() function except that the terminating '\n' is removed. The following statements illustrate a use of wgets():

```
char buf[81]; int retcod;
retcod = wgets(buf,15);
switch (retcod)
{
    case 1: wputs("\nBREAK detected");
            break;
    case 2: export();
            exit();
}
```

The wgets() function sequence will invoke WKEYIN. A blinking cursor will appear in the window and you will be permitted the input. This function does not echo CR's, does not accept tabs, nor any character less than X'20' <SPACE> or greater than X'7F' <DELETE>.

If the return code is a zero (0), no special action need be taken. However, if it is either one (1) or two (2), then your program may want to take a particular action since either <BREAK> was pressed or EXPORT was requested. You can establish code for either alternative to handle the result. A value of two (2) implies that the responder wanted to EXPORT. This request is satisfied under program control via the export() function. the code value of one (1) lets you deal with the <BREAK> key within your program. The example illustrates these choices via a switch-case construct.

The wgets() function itself is coded in the library as follows:

```
wgets(buf,max)
char *buf; int max;
{   char *regs[6];
    regs[BC] = max & 0xff;
    regs[HL] = buf;
    call(124,regs);
    *(++buf + (regs[BC] >> 8)) = '\0';
     if ((regs[AF] & 1) == 0)
        return(0);
    else
        return((regs[BC] & 0xff) + 1);
}
```

Window "Peek" - Function wpeek()

This function will peek the character at the given window location. It requires parameters of window row and window column. The actual character peeked is returned as the function's value. The following statements illustrate a use of wpeek():

```
int c,row, col;
row = 3; col = 7;
c = wpeek(row,col);
```

The function itself is coded in the library as follows:

```
wpeek(row,col)
int row,col;
{   char *regs[6];
    regs[BC] = 0x100;
    regs[HL] = row << 8 | col;
    call(124,regs);
    return(regs[AF] >> 8);
}
```

Window "Poke" - Function wpoke()

This function will poke a character into the window at the given window location. It requires parameters of window row, window column, and character value. The following statements illustrate a use of wpoke():

```
int row, col;
row = 2;
for (col = 0; col < 20; col++)
    wpoke(' ',row,col);
```

In this example, window row 2 is being blanked from column 0 to 19 because the character value passed is a BLANK.

The function itself is coded in the library as follows:

```
wpoke(c,row,col)
int c,row,col;
{   char *regs[6];
    regs[BC] = 0x200 | (c & 0xff);
    regs[HL] = row << 8 | col;
    return(call(124,regs));
}
```

Set the Window Cursor Position - Function wscur()

This function will establish the window cursor's position to a program selected coordinate. The function requires parameters of window row and window column. The following statements illustrate a use of wscur():

```
int row, col;
row = col = 0;
wscur(row,col);
```

This will position the cursor at the beginning of the window (upper left corner of the window) which is row-0, column-0.

The function itself is coded in the library as follows:

```
wscur(row,col)
int row,col;
{   char *regs[6];
    regs[BC] = 0x300;
    regs[HL] = row << 8 | col;
    return(call(124,regs));
}
```

Obtain the Window Cursor Position - Function wgcur()

This function is used to obtain the row and column position of the cursor within the window. The actual values of row and column are returned as the function value (row is the high order, column is the low order). The following statements illustrate a use of wgcur():

```
int rowcol, row, col;
rowcol = wgcur();
col = rowcol && 0xff;
row = rowcol >> 8;
```

The function itself is coded in the library as follows:

```
wgcur()
{   char *regs[6];
    regs[BC] = 0x400;
    call(124,regs);
    return(regs[HL]);
}
```

Buffer to Window - Function wb2w()

This driver function will transfer a video image contained in a block of RAM to the window. Since the image will be copied into the entire window, the block of RAM must be at least equal to the size of the window (rows x columns). Such a block may be defined in C by establishing a character array of length equal to the size of the window. It is probably more suitable to define the array as static. The following statements illustrate a use of wb2w():

```
char buffer[200];
wb2w(buffer);
```

The function itself is coded in the library as follows:

```
wb2w(buffer)
char *buffer;
{   char *regs[6];
    regs[BC] = 0x500;
    regs[HL] = buffer;
    return(call(124,regs));
}
```

Window to Buffer - Function ww2b()

This function will copy the window image to a block of RAM. Since the entire window will be copied to RAM, the block of RAM must be at least equal to the size of the window (rows x columns). Such a block may be defined in C by establishing a character array of length equal to the size of the window. It is probably more suitable to define the array as static. The following statements illustrate a use of ww2b():

```
char buffer[200];
ww2b(buffer);
```

The function itself is coded in the library as follows:

```
ww2b(buffer)
char *buffer;
{   char *regs[6];
    regs[BC] = 0x600;
    regs[HL] = buffer;
    return(call(124,regs));
}
```

Create a Window - Function wcreat()

This function will create a new window. If the maximum depth level has been reached prior to the request, the function will return with a value of one (1) which indicates the error code. A non-error condition is returned as NULL. The following statements illustrate a use of wcreat():

```
if (wcreat(5,5,10,20) != NULL)
    {fputs("Can't open the window\n",stderr); exit(-1);}
```

In this example, PRO-NTO creates a window of 10 columns and 20 rows. The north west corner of the window is positioned at row-5, column-5 on the screen. Thereafter, all references to the window locations are relative to 0,0. Thus, window row numbers range from 0 through 9 while column numbers range from 0 through 19. Since your window does not exceed 22 rows by 78 columns, PRO-NTO automatically draws a box around the window.

The function itself is coded in the library as follows:

```
wcreat(srow,scol,nrows,ncols)
int srow,scol,nrows,ncols;
{   char *regs[6];
    regs[BC] = 0x700;
    regs[DE] = nrows << 8 | ncols;
    regs[HL] = srow << 8 | scol;
    return(call(124,regs));
}
```

Close a Window - Function wclose()

This C function is used to close the last window opened via wcreat(). Any program which opens a window must close it before terminating. Also, if you have opened more than one window, you can only return to a previous window by closing the current one. The following statement illustrates a use of wclose():

```
wclose();
```

The function itself is coded in the library as follows:

```
wclose()
{   char *regs[6];
    regs[BC] = 0x800;
    call(124,regs);
}
```

Display a Character in the Window - Function wputchar()

This function will display a character in the window at the current window cursor location then advance the cursor. It is similar to C's putchar() function. The window display driver supports the following control codes:

CTL	Value	Function
BKSP	0x08	Backspace and erase.
LF	0x0a	Linefeed.
CR	0x0d	Carriage return and linefeed.
LEFT	0x18	Move cursor left.
RIGHT	0x19	Move cursor right.
DOWN	0x1a	Move cursor down.
UP	0x1b	Move cursor up.
HOME	0x1c	Home cursor to position 0,0.
START	0x1d	Home to column zero.
EREOL	0x1e	Erase to end of line.
EREOW	0x1f	Erase to end of window.

Any control code not supported by the window driver will be passed through to the DOS video driver. The following statements illustrate a use of the function, wputchar():

```
puta2t()
{   int i;
    wscur(2,0);                          /* Position cursor */
    for (i=0; i<20; i++)
        wputchar(i+65);                      /* Print A to T */
}
```

The function itself is coded in the library as follows:

```
wputchar(c)
int c;
{   char *regs[6];
    regs[BC] = 9 << 8 | (c & 0xff);
    call(124,regs);
}
```

Display a String in the Window - Function wputs()

This function will be similar to puts() except that the target is the window rather than the display device. The following statement illustrates a use of wputs():

```
wputs("This is a sample message\nwhich is two lines long\n");
```

The function itself is coded in the library as follows:

```
wputs(string)
char *string;
{   char c;
    while (c=*string++)
        wputchar(c);
}
```

Window Export - Function wexport()

This function is used to support exporting of data from the current window back to the previous screen. Export may be a user response to a wgets() at which point the program decides whether or not to satisfy that request via invoking this function. Export may also be invoked purely at the discretion of the program. Remember, EXPORT will automatically close the current window and enter the PRO-NTO marking facility. Once the line or block is marked by the user, that marked data will be fed back to the previous screen during gets(), getchar(), or wgets() requests. The following statements illustrate a use of wexport():

```
char buf[81];
switch (wgets(buf,15))
{
    case 1: wputs("\nBREAK detected");
            break;
    case 2: export();
            exit();
```

The function itself is coded in the library as follows:

```
wexport()
{   char *regs[6];
    regs[BC] = 0x801;
    call(124,regs);
}
```

Window Import - Function wimport()

This function is used to perform the operation of IMPORT. When invoked, the previous screen will be restored temporarily and the IMPORT cursor will be presented. After the block of text is marked, PRO-NTO will restore the application's window and return from the function. Thereafter, invocations of wgets() or other C input requests will be satisfied from the marked text until the text is exhausted, at which time key requests will revert to the physical keyboard. The complete C program example illustrates wimport().

The wimport() function itself is coded in the library as follows:

```
wimport()
{   char *regs[6];
    regs[BC] = 0xb00;
    call(124,regs);
}
```

Complete C Program Example

The following sample program illustrates a number of the functions which have just been discussed. Essentially, the program creates a window, copies the initial blanked screen to a character array RAM block, displays 25 numbered lines of text, displays the characters "A" through "T" on the third line, waits then blanks the line, requests a keyin response, blanks the window by copying the saved window back to the window, closes the window and issues a gets() so that it becomes satisfied by window export or other keyboard input. The astute reader may note that this program in C develops a process which is identical to the process developed by the WINLINK/BAS program example using BASIC.

```
#include stdio/csh
#option INLIB
#option REDIRECT OFF
main()
{   int i, row, col, retcod, rowcol;
    char buffer[200], keybuf[81];
    if (wcreat(5,5,10,20) != NULL)
    {fputs("Can't open the window\n",stderr); exit(-1);}
    ww2b(buffer);
    for (i = 1; i < 26; i++)
    {   wputs("This is test ");
        itoa(i,keybuf);
        wputs(keybuf);
        wputchar('\n');
    }
    wait();
    puta2t(); wait();
    rowcol = wgcur(); col = rowcol & 0xff; row = rowcol >> 8;
    blank(); wait();
    retcod = wgets(keybuf,10);
    switch (retcod)
    {   case 1: wputs("\nBREAK detected");
                break;
        case 2: wexport();
                break;
    }
    if (retcod != 2)
    {    wb2w(buffer); wait();
         wclose();                    /* Close the window */
```

```
    }
   printf("Row = %d, Col = %d\n",row, col);
   printf("Retcod = %d, Keyin = %s\n",retcod,keybuf);
   puts("By your command: "); gets(keybuf);
   puts(keybuf); putchar('\n');      /*  Do input (any export?) */
}
puta2t()
{   int i;
    wscur(2,0);                     /* Position cursor */
    for (i=0; i<20; i++)
        wputchar(i+65);                 /* Print A to T */
}
blank()
{   int i;
    for (i=0; i<20; i++)
        wpoke(' ',2,i);
}
wait()
{   int i;
    for (i = 0; i < 10000; i++);
}
#include window/ccc
```

What is WINLINK?

The PRO-NTO window manager is controlled via a DOS SuperVisor Call (SVC) facility which is attached to the resident DOS when PRO-NTO is installed. Access to the SVC is most normally done by machine language routines; however, the window manager may be controlled from any programming language. If you are programming in "C", you may find it convenient to control windows by utilizing the C-function library provided with the PRO-NTO package. If, on the other hand, you are programming in "BASIC", you may find it very inconvenient to write interfacing assembly language routines to interface with BASIC's USR or CALL facility. Because of this inconvenience, the WINLINK driver has been provided with the PRO-NTO package to facilitate the control of the window via a series of PRINT# and INPUT# statements.

WINLINK installs a system device driver that can be OPENed from BASIC (or any programming language which permits the opening of a file). After it is opened for sequential input and output (via OPEN statements), you can easily control all aspects of the window environment via INPUT# and PRINT# statements. The following sections describe the installation and use of WINLINK. Remember, in order to gain access to the PRO-NTO window manager via the WINLINK driver, you must have already installed PRO-NTO itself!

Installing WINLINK/CMD

The WINLINK device driver is easily installed by typing the command:

WINLINK

WINLINK will first display an appropriate welcome message such as:

WINLINK - PRO-NTO Window SVC Linkage Driver Version 1.0a
Copyright (c) 1985 by MISOSYS, Inc., All rights reserved

It will then check to make sure that 1) PRO-NTO has already been installed, and 2) that WINLINK has not already been installed. If it cannot locate the PRONTO window manager, it will display the error message:

Please install PRO-NTO first!

and abort your request. If it locates the WINLINK device in the system, it will display the error message:

The *WL device is already in use!

and abort the request. If such is not the case, WINLINK will then locate a spare system Device Control Block (DCB). If it cannot obtain the DCB from the system, it will display the error message,

No system DCB is available for *WL!

Once it obtains the DCB, it will locate sufficient memory space in high memory. If it cannot do this because high memory alteration is restricted, it will display the error message:

Can't alter high memory pointer!

Once it obtains the needed memory, it will relocate its device driver to that high memory space, and insert the necessary information into the DCB established for the device. It will display the completion message:

***WL device is now installed**

Note that the name of the device is "*WL". This will be the "filename" used in BASIC's OPEN statement. Remember, the DOS permits you to OPEN devices, such as "*PR", "*DO", etc., as well as actual files. The installation of WINLINK is now complete; you may begin using it in a BASIC program, a FORTRAN program, a PASCAL program, or other such language that permits sequential file input and output.

Removing the WINLINK device driver

Once WINLINK is installed, it may be removed from the Device Control Block table and from high memory - provided it was the last module installed in high memory. This removal is achieved via the command:

WINLINK (REMOVE)

In general, the closing parenthesis is not required. Also, the parameter, "REMOVE", may be abbreviated to the single letter, "R". The word, "OFF", may also be entered in lieu of "REMOVE" or "R". If you enter this parameter incorrectly, or enter some other un-supported parameter, the following error message will be displayed and the program will abort.

Parameter error!

Under this command request, WINLINK first checks to make sure that the WINLINK device driver is actually already installed. If it is not, WINLINK will display the error message:

The *WL device is not installed!

and abort the request. If it is installed, but the current high memory pointer maintained by the system does not point to the memory location immediately preceding the WINLINK driver (this means that some other module was installed into high memory after you installed WINLINK), WINLINK cannot "unhook" itself. It will thusly display the error message:

Can't reclaim high memory - *WL device not removed!

The only other case is where the high memory module will be removed and the message informing you of this will be presented as follows:

***WL device is now removed**

Controlling the window manager

The remaining sections will show you how to control the window manager so that you can create windows and use them. So that we can make the explanation "universally" understandable, all examples will be illustrated in BASIC statements. Bear in mind, though, that the *WL device driver can be used from any language.

Languages generally associate a file or device name with some unit number. In BASIC, this unit is called the file buffer. Since the device driver is a sequential two-way driver, in BASIC, we would open it for both input and output. An appropriate set of statements would be:

```
10 OPEN "O",1,"*WL": OPEN "I",2,"*WL"
```

In this example, file buffer 1 is assigned to output (achieved via PRINT#1 statements) while file buffer 2 is assigned to input (achieved via INPUT#1 statements). The OPEN statements will, most likely, be the first statements of your program (after variable setup).

WINLINK supports twelve functions of communications with the device driver. These functions are: keyin input, peek, poke, set cursor position, obtain cursor position, copy image buffer to window, copy window to image buffer, create a window, close a window, display a character or string, export a block, and import a block; all functions referencing operation on the window. To invoke a function, the program must output a start-of-header (SOH) character, followed by the function number. If the function requires parameters, they must then be output as a series of single byte(s). If the function provides input, follow up the function output request with an INPUT# statement. All parameters and codes output to the device via the PRINT# statement must be character values (single bytes). These bytes are generally output via the "CHR$(var)" function in BASIC. The following sections explain the use of the twelve functions.

WINDOW KEYIN - Function 0

This KEYIN function is used to perform a line input while blinking the cursor (so you can tell where it is if you have not otherwise turned it on). It is somewhat like BASIC's INPUT statement except that WINDOW KEYIN will always blink the window cursor and display the character input. The function will pass back status concerning the depression of <ENTER>, <BREAK>, or EXPORT. This means that a WKEYIN request must be followed by an INPUT# request. The first variable returned will be an integer value of zero, one, or two if respectively <ENTER>, <BREAK>, or EXPORT terminated the keyed in

response. The second variable would be a string variable if your input was to be interpreted as a string. It could also be a series of integer variables if the input response was to be a series of numeric values. If IMPORT is keyed during the key in response, PRO-NTO will automatically handle the request within the WKEYIN processing.

When you invoke WKEYIN, you must provide a count of the maximum number of characters permitted for input as a parameter. This is different from BASIC's INPUT verb where no count is specified but is highly desirable in the case of windows. The count can range from 1 to 79. The following statements illustrate a use of WKEYIN:

```
400 PRINT#1,CHR$(1);CHR$(0);CHR$(10);:INPUT#2,FLAG,A$
```

Note that we are outputing a SOH [the CHR$(1)] to initiate a funtion request. This is followed by the function number [CHR$(0)], then the maximum number of characters permitted for input [CHR$(10)], which is ten characters in this case. Also, note that each CHR$(VAR) is suffixed with a semi-colon. The semi-colon is used to tell BASIC not to output a series of blanks after the character which BASIC would normally do to establish field positions. That particular output sequence will invoke WKEYIN. A blinking cursor will appear in the window and you will be permitted the input. Unlike BASIC's INPUT verb routine, this function does not echo CR's, does not accept tabs, nor any character less than X'20' <SPACE> or greater than X'7F' <DELETE>.

Note, though, that what is typed at the keyboard will not be passed back to the program until the INPUT# statement is executed. If the variable, FLAG, contains a zero (0), no special action need be taken. However, if it is either one (1) or two (2), then your program may want to take a particular action since either <BREAK> was pressed or EXPORT was requested. You can establish code for either alternative and use an ON FLAG GOTO or ON FLAG GOSUB to handle the result. A FLAG value of two (2) implies that the responder wanted to EXPORT. This request is satisfied under program control via the EXPORT function (see function 10). If the responder depressed the <BREAK> key, this will NOT terminate the BASIC program. Thus, the FLAG value of one (1) lets you deal with the <BREAK> key within your program.

WINDOW "PEEK" - Function 1

This function will peek the character at the given window location. It requires parameters of window row and window column. The actual character peeked is obtained via an INPUT#. The following statements illustrate a use of WPEEK:

```
PRINT#1,CHR$(1);CHR$(1);:INPUT#2,A$
```

The input variable [A$ in this example] will be a one-character string after the INPUT#2 is executed.

WINDOW "POKE" - Function 2

This function will poke a character into the window at the given window location. It requires parameters of window row, window column, and character value. The following statements illustrate a use of WPOKE:

```
350 FOR I = 0 TO 19
360 PRINT#1,CHR$(1);CHR$(2);CHR$(2);CHR$(I);" ";
370 NEXT
```

In this example, window row 2 is being blanked from column 0 to 19. The character value is passed as a one-charml[l[string containing a BLANK character. You see, a one-character string as shown produces the same value as "CHR$(32)".

SET THE WINDOW CURSOR POSITION - Function 3

This function will establish the window cursor's position to a program selected coordinate. The function requires parameters of window row and window column. The following statement illustrates a use of WSCUR:

```
310 PRINT#1,CHR$(1);CHR$(3);CHR$(0);CHR$(0);
```

This will position the cursor at the beginning of the window (upper left corner of the window) which is row-0, column-0.

OBTAIN THE WINDOW CURSOR POSITION - Function 4

This function is used to obtain the row and column position of the cursor within the window. The actual values of row and column must be obtained via an INPUT# statement. The following statements illustrate a use of WGCUR:

```
165 PRINT#1,CHR$(1);CHR$(4);:INPUT#2,WROW,WCOL
```

Make sure you don't try to use "COL" as the variable for column as "COL" is a reserved word in BASIC.

BUFFER TO WINDOW - Function 5

This driver function will transfer a video image contained in a block of RAM to the window. Since the entire window will be copied into, the block of RAM must be at least equal to the size of the window (rows x columns). Such a block may be captured in BASIC by establishing a single dimension integer array of length equal to half the size of the window (integers occupy two bytes per element). The MAJOR CAUTION to be observed is that BASIC will reposition arrays if scalers are declared after the array declaration. For this reason, it is important to ALWAYS declare all of your scaler variables

(undimensioned integers, single precisions, double precisions, and strings) prior to the integer array dimension. The following statements illustrate a use of WB2W:

```
6 AL=0:AH=0:I=0:WROW=0:WCOL=0:FLAG=0: A$=""
10 OPEN "O",1,"*WL": OPEN "I",2,"*WL"
25 DIM BUF%(100):ADDR = VARPTR(BUF%(0)):AL=ADDR AND 255:
   AH=CINT(ADDR/256) AND 255
100 PRINT#1,CHR$(1);CHR$(7);CHR$(5);CHR$(5);CHR$(10);CHR$(20);
... more statements ...
190 PRINT#1,CHR$(1);CHR$(5);CHR$(AH);CHR$(AL);
```

In this illustration, all of the scalers used by the program have been declared in statement 6. After the *WL device is opened, the address of the first array element is determined [via VARPTR(BUF%(0))]. This address is converted to the high order byte (AH) and low order byte (AL) via the remaining statements in line 25. Line 100, of course, creates a window of 10 rows and 20 columns. Additional statements could poke text into the integer array for subsequent moving to the window. The integer array block could also have received an image from the window itself (see the example in WW2B, function 6). In line 190, the WB2W function is invoked wherein PRO-NTO will copy the contents of your array (the RAM block) into the window.

WINDOW TO BUFFER - Function 6

This function will copy the window image to a block of RAM. Since the entire window will be copied, the block of RAM must be at least equal to the size of the window (rows x columns). Such a block may be captured in BASIC by establishing a single dimension integer array of length equal to half the size of the window (integers occupy two bytes per element). The MAJOR CAUTION to be observed is that BASIC will reposition arrays if scalers are declared after the array declaration. For this reason, it is important to ALWAYS declare all of your scaler variables (undimensioned integers, single precisions, double precisions, and strings) prior to the integer array dimension. The following statements illustrate a use of WW2B:

```
6 AL=0:AH=0:I=0:WROW=0:WCOL=0:FLAG=0: A$=""
10 OPEN "O",1,"*WL": OPEN "I",2,"*WL"
25 DIM BUF%(100):ADDR = VARPTR(BUF%(0)):AL=ADDR AND 255
   :AH=CINT(ADDR/256) AND 255
100 PRINT#1,CHR$(1);CHR$(7);CHR$(5);CHR$(5);CHR$(10);CHR$(20);
105 PRINT#1,CHR$(1);CHR$(6);CHR$(AH);CHR$(AL);
```

In this illustration, all of the scalers used by the program have been declared in statement 6. After the *WL device is opened, the address of the first array element is determined [via VARPTR(BUF%(0))]. This address is converted to the high order byte (AH) and low order byte (AL) via the remaining statements in line 25. Line 100, of course, creates a window of 10 rows and 20 columns. In line 105, the WW2B function is invoked wherein PRO-NTO will copy an image of the window into your array.

CREATE A WINDOW - Function 7

This device function will create a new window. If the maximum depth level has been reached prior to the request, the function will return with an error code. Since BASIC generally ignores error codes returned from DEVICE I/O, this error can not be tested via an ON ERROR GOTO statement. Other programming languages probably will not ignore the other. Because BASIC does, and because the status returned from WCREAT is very important, the return code is also set up as a value to be input via INPUT#. The following statements illustrate a use of WCREAT:

```
100 PRINT#1,CHR$(1);CHR$(7);CHR$(5);CHR$(5);CHR$(10);CHR$(20);
101 INPUT#2,FLAG:IF FLAG = 0 THEN GOTO 105:
    PRINT"Can't open window":STOP
```

In this example, PRO-NTO creates a window of 10 columns and 20 rows. The north west corner of the window is positioned at row-5, column-5 on the screen. Thereafter, all references to the window locations are relative to 0,0. Thus, window row numbers range from 0 through 9 while column numbers range from 0 through 19. Since your window does not exceed 22 rows by 78 columns, PRO-NTO automatically draws a box around the window.

Since the programming language used is BASIC, immediately following the WCREAT request is an INPUT#. This will then read the return code as an integer value. If the window was successfully created, the value will be zero. Any other value will indicate an error.

CLOSE A WINDOW - Function 8

This driver function is used to close the last window opened via WCREAT. Any program which opens a window must close it before terminating. Also, if you have opened more than one window, you can only return to a previous window by closing the current one. The following statement illustrates a use of WCLOSE:

```
200 PRINT#1,CHR$(1);CHR$(8);
```

DISPLAY A CHARACTER IN THE WINDOW - Function 9

This function will display a character in the window at the current window cursor location then advance the cursor. It is similar to BASIC's PRINT statement, not WPOKE! The window display driver supports the following control codes:

CTL	Value	Function
BKSP	8	Backspace and erase.
LF	10	Linefeed.
CR	13	Carriage return and linefeed.
LEFT	24	Move cursor left.
RIGHT	25	Move cursor right.
DOWN	26	Move cursor down.
UP	27	Move cursor up.
HOME	28	Home cursor to position 0,0.
START	29	Home to column zero.
EREOL	30	Erase to end of line.
EREOW	31	Erase to end of window.

Any control code not supported by the window driver will be passed through to the DOS video driver. The following statements illustrate a use of WDSP:

```
110 PRINT#1,CHR$(1);CHR$(9);
120 FOR I = 1 TO 25
130 PRINT#1,"This is test ";I
140 NEXT
```

Note in this example that the SOH-FUNCTION is output only once for the entire printing loop. Output of the display function sets up the device driver so that all subsequent character values are treated as characters to display until another SOH is detected. This method makes it very easy to display images just like you would on the regular screen using PRINT. Since BASIC formats the output from PRINT# statements identically to PRINT statements, you need only learn to code the numbersign into your programs after you output the SOH and function-9.

WINDOW EXPORT - Function 10

This device driver function is used to support exporting of data from the current window back to the previous screen. Export may be a user response to a WKEYIN at which point the program decides whether or not to satisfy that request via invoking this function. Export may also be invoked purely at the discretion of the program. Remember, EXPORT will automatically close the current window and enter the PRO-NTO marking facility. Once the line or block is marked by the user, that marked data will be fed back to the previous screen during INPUT (or WKEYIN) requests. The following statement illustrates a use of WEXPORT:

```
180 GOSUB 400: ON FLAG GOSUB 410,420
... more statements ...
400 PRINT#1,CHR$(1);CHR$(0);CHR$(10);
    :INPUT#2,FLAG,A$:RETURN
410 RETURN
420 PRINT#1,CHR$(1);CHR$(10);:RETURN
```

In this example, a WKEYIN response flag is being tested by an ON ... GOSUB statement. The subroutine numbered 420 is executed in the case of a user's EXPORT response.

WINDOW IMPORT - Function 11

This device driver function is used to perform the operation of IMPORT. When invoked, the previous screen will be restored temporarily and the IMPORT cursor will be presented. After the block of text is marked, PRO-NTO will restore the application's window and return from the function. Thereafter, invocations of WINDOW KEYIN or other BASIC input requests will be satisfied from the marked text until the text is exhausted, at which time key requests will revert to the physical keyboard.

Complete BASIC program example

The following sample program illustrates a number of the functions which have just been discussed. Essentially, the program creates a window, copies the initial blanked screen to a integer array RAM block, displays 25 numbered lines of text, displays the characters "A" through "T" on the third line, waits then blanks the line, requests a keyin response, blanks the window by copying the saved window back to the window, closes the window and issues an INPUT so that it becomes satisfied by window export or other INPUT.

```
1 REM WINLINK/BAS - 03/22/85 - WINLINK interface example by MISOSYS, Inc.
6 AL=0:AH=0:I=0:WROW=0:WCOL=0:FLAG=0: A$="": REM Declare all variables
10 OPEN "O",1,"*WL": OPEN "I",2,"*WL": REM Open *WL for input/output
25 DIM BUF%(100):ADDR = VARPTR(BUF%(0)):AL=ADDR AND 255
   :AH=CINT(ADDR/256) AND 255: REM Get pointer to RAM block
100 PRINT#1,CHR$(1);CHR$(7);CHR$(5);CHR$(5);CHR$(10);CHR$(20);
101 INPUT#2,FLAG:IF FLAG = 0 THEN GOTO 105:PRINT"Can't open window":STOP
105 PRINT#1,CHR$(1);CHR$(6);CHR$(AH);CHR$(AL);
110 PRINT#1,CHR$(1);CHR$(9);: REM Initiate display mode
120 FOR I = 1 TO 25
130 PRINT#1,"This is test ";I
140 NEXT
150 FOR I = 1 TO 1000: NEXT: REM Pause
160 GOSUB 300: GOSUB 999: REM Put A-T then pause
165 PRINT#1,CHR$(1);CHR$(4);:INPUT#2,WROW,WCOL:REM Get cursor position
170 GOSUB 350: GOSUB 999: REM Blank the A-T line
180 GOSUB 400: ON FLAG GOSUB 410,420: REM Get keyin, act on response
190 PRINT#1,CHR$(1);CHR$(5);CHR$(AH);CHR$(AL);:GOSUB 999
200 PRINT#1,CHR$(1);CHR$(8);: REM Close the window
205 PRINT WROW,WCOL: REM Print result of line 165
206 PRINT "Flag = ";FLAG,A$: REM Print the response of line 180
207 INPUT"By your command: ";A$:PRINT A$: REM Do input (any export?)
210 END
300 PRINT#1,CHR$(1);CHR$(3);CHR$(2);CHR$(0);: REM Position cursor
310 PRINT#1,CHR$(1);CHR$(9);: REM Initiate display mode
320 FOR I = 0 TO 19: PRINT#1,CHR$(I+65);: REM Print A to T
330 NEXT
340 RETURN
350 FOR I = 0 TO 19: REM Blank row 2
360 PRINT#1,CHR$(1);CHR$(2);CHR$(2);CHR$(I);" ";
370 NEXT
380 RETURN
400 PRINT#1,CHR$(1);CHR$(0);CHR$(10);:INPUT#2,FLAG,A$:RETURN
410 RETURN
420 PRINT#1,CHR$(1);CHR$(10);:RETURN
999 FOR I = 1 TO 2000: NEXT: RETURN
1000 OPEN "O",1,"*WL":PRINT#1,;CHR$(1);CHR$(8);:END: In case of STOP
9999 SAVE"winlink/bas:6: REM Save the program
```

PRO-NTO - A Window and Application Manager
Appendix - List of Files

The following files are supplied with PRO-NTO:

File Specification		Purpose of file
ADDRESS/APP		The ADDRESS mail and rotating index card application
ADDRESS/DAT		A sample address data file
AFPCALC/APP	*	The Algebraic Floating Point CALCulator application
BRINGUP/APP		The BRINGUP tickler and appointment book application
BRINGUP/BAS		A program to read and display a BRINGUP data file
CAL/APP	*	The perpetual CALENDAR application
CARD/APP	*	The CARD file and notepad application
CEXAMPLE/CCC		A program illustrating use of WINDOW/CCC
CHARSET/APP		The CHARacter SET application
DEFAULTS/CMD		A program to alter PRONTO's application defaults
DIALER/APP	*	The telephone list and auto DIALER application
DIALER/DAT		A sample dialer telephone list
DOSAVE/APP		The DOSAVE application
DOSAVE/ASM		The PRO-CREATE assembler source file to DOSAVE/APP
HELPP/CMD		A help-screen facility for PRO-NTO
PRONTO/CMD	*	Installs the window and application manager
PRUN/CMD		The PRUN facility
PSORT/CMD		Utility to sort application data files
RPNCALC/APP		The Reverse Polish Notation CALCulator application
README/TXT		May contain last minute documentation - Read it!
TERM/APP		The mini-TERMinal application
TYPER/APP		The TYPE writeR application
TYPER/ASM		The PRO-CREATE assembler source file to TYPER/APP
WINDOW/CCC		The PRO-LC interface C library
WINDOW/MAC		Macro library of @WINDOW SVC calls
WINLINK/BAS		A program illustrating use of *WL
WINLINK/CMD		The *WL SVC interface device driver

In order to begin with PRO-NTO in a two-drive system, just use one of your working BACKUP copies in the top drive. If you want to only transfer a minimum PRO-NTO set of files to your SYSTEM disk or if you have only a one-drive system, at a bare minimum you will need to copy to your SYSTEM drive all of the files appended with an asterisk in the table above. Once you get more familiar with the use of each of the files provided with PRO-NTO, you will be in a better position to choose your file set.

DW2PS/FLT - A DW-II Proportional Space Filter.
LCOPY - Transfer DOS files to CP/M+
PRO-ADE - A facility to provide a two-tiered directory structure. Ideal for the hard disk user or other large drive user.
PRO-CESS - A load module maintenance utility.
PRO-CON80Z - Translate 8080 files to Z-80.
PRO-CREATE - Macro assembler supports nested macros, includes, and conditionals. Includes full screen text editor and X-reference.
PRO-CURE - Transfer files from 19 different CP/M formats to TRSDOS
PRO-DD&T - Adds a Disassembler to your DEBUG display. Almost makes debugging apleasure. Includes a dynamic trace facility.
PRO-DESCRIBE - Know the contents of your files! Extends your directory with a 63 character descriptor for each file. Custom dir command.
PRO-DUCE - Two pass labeling disassembler. From disk or memory; to screen, printer, or disk.
PRO-ESP - Altdisk, Altld, Altres, Ded, CRLF,Ctlg, Cvt324, Doedit, Fkey, Iomon, MiniDOS, Name, PRtoggle, Rd40, Unremove, Xonxoff
PRO-GENY - DOCONFIG to save/restore configs, MEMDIR to get memory dirs, PARMDIR for JCL generator, and SWAP your DCT's.
PRO-HartFORTH - A full 79-Standard FORTH compiler which runs under the DOS.
PRO-HELP - A Help facility used to generate our HELP files [HELPP].
PRO-IFC - Interactive File Controller. Lets you copy, rename, delete, or list files from menu. Wilcard tagging. Easy to use.
PRO-LC - Integer subset C compiler with substantial function library.
PRO-MACH2 - File Allocation utility. You control where files are placed. With Mapper, Alloc, Calc, and Handy.
PRO-MLIB - A librarian for /IRL and /REL files.
PRO-PaDS - Partitioned Data Set utility. Invoke /CMD files directly. Save disk space by collecting short files into one.
PRO-SAID - Full Screen text editor great for C, ASM, and other languages.
PRO-XFTS - X-modem protocol file transfer system - works from JCL.
PRO-ZCAT - Catalog disks of files with this easy to use and fast tool.
PRO-ZGRAPH - Pixel graphic screen editor prints to Epson and DMP printers.
PRO-ZSHELL - Add command line I/O redirection, piping, and multiple commands on a line to your DOS. UNIX-like features on your Mod 4!
PROGRAMMER'S GUIDE TO TRSDOS 6.x by Roy Soltoff is the definitive answer to interfacing the DOS. Six chapters with three complete filters.